MODCOMP
MODULAR COMPUTER SYSTEMS, INC.

MAX IV System Guide Manual

Executive (REX) Services

SOFTWARE

MODULAR COMPUTER SYSTEMS, INC

MAX IV System Guide Manual

Executive (REX) Services

REVISION INSTRUCTIONS

Revision I01, July 1985
MAX IV SYSTEM GUIDE MANUAL,
EXECUTIVE (REX) SERVICES
213-804003-I01

To update your manual, insert the enclosed pages according to the instructions:

Remove:	Insert:
Cover/Blank	Cover/Blank
Title/Blank	Title/Blank
iii/Blank	iii/(iv blank)
vii/viii	vii/viii
ix/x	ix/x
xi/xii	xi/xii
23/24	23/24
25/26	25/26
27/28	27/28
29/30	29/29A 30/(30A blank)
31/32	31/32
37/38	37/38
41/42	41/42
43/44	43/44
103/104	103/104
105/106	105/106
109/110	109/110
137/38	137/138

NOTE: Revised pages are marked with revision bars and the corresponding revision level. When the manual is reissued, all outstanding revisions will be incorporated.

MANUAL HISTORY

Manual Order Number: 213-804003-I01

Title: MAX IV SYSTEM GUIDE MANUAL, EXECUTIVE (REX) SERVICES

Software Part Number: 610304-000I.1

Model Number: 8240I

Revision Level	Date Issued	Description
H00	03/83	Initial Issue (H.0). This manual replaces Chapter 3 of manual order number 220-610304-000 and Chapter 7 of manual order number 210-610304.
H01	11/83	Revision (H.1).
H02	05/84	Reissue (H.2).
I00	09/84	Reissue (I.0).
I01	07/85	Revision (I.1). Changes were made to the following services: ASSIGN, TASSIGN, WAIT, ABORT, LOVER, USERTRAP, 'SYSCON' format of GETSYS, GETTIME, TIME.

Contents subject to change without notice.

PREFACE

Audience: The information in this manual is directed to the applications programmer or developer with a good knowledge of MODCOMP Assembler language and familiarity with MAX IV SYSGEN.

Subject: This manual contains information about the MAX IV Executive (REX) Services. The information includes a description of the service performed, the Assembly language calling sequence, error conditions (if any) and an explanation of returned values (if any). Where applicable, examples and programming considerations are given to help the programmer interface with the operating system.

Information about the theory of operation of the REX services and the unimplemented instruction trap of MAX IV is presented in Chapter 4. The processing of unimplemented instructions, including the REX instruction and the floating point instruction set is described. This information is extended to describe how MAX IV implements executive service subroutines using the REX mechanism. Techniques for coding custom REX services are also described.

Those users familiar with previous issues of MAX IV Operating Systems manuals may notice that the information in this manual appeared in either:

Chapter 7 MAX IV GENERAL OPERATING SYSTEM Reference Manual
Chapter 3 MAX IV GENERAL OPERATING SYSTEM Technical Manual

Products Supported: MAX IV Software System, Model 8240I.

Related Publications: The reader is referred to the manuals listed below for additional information. When ordering manuals, use the Manual Order Number listed in this Preface. The latest revision (REV) will be shipped.

Manual Order Number	Manual Title
213-804001-REV	MAX IV GENERAL OPERATING SYSTEM System Guide Manual
213-804002-REV	MAX IV SYSGEN System Guide Manual
213-804005-REV	MAX IV BASIC I/O SYSTEM System Guide Manual
211-804011-REV	MAX IV TASK/OVERLAY CATALOGER (TOC) Programmer's Reference Manual
215-804004-REV	MAX IV UTILITY LIBRARY Library Reference Manual
210-600303-000	MAX III GENERAL OPERATING SYSTEM Reference Manuals

Special Symbols and Notations: A revision bar (|) located in the margin of the page in the text indicates a change to the manual. This change generally represents a technical change to the product due to product revision. A revision bar is also entered in the Table of Contents to flag the general location of changes in the text.

MAX IV REVISION I.1 SUMMARY

ASSIGN Service

The order of precedence in searching for a referenced entity "Y" (as in ASSIGN X Y) has been changed. The order is to first search the task's FAT. If the named entity is not found there, the system's global FAT is then searched, followed by the system device list. The DEVPRE option was added to the ASSIGN service to use the previous searching precedence (system device list, task's FAT, global FAT). The K-task now uses the DEVPRE option to retain the integrity of logic used to resolve assignments for resident tasks during a cold start in two passes.

TASSIGN Service

Bit 7 of Register 8 must be set if Register 9 is to be used as an extended option register. Additional options are specified in Bits 2 and 3 of Register 9.

By setting a bit in the Extended Option Word of the UPT, a privileged calling task can use the logical file assignments of another task when referencing that task's FAT.

WAIT and ABORT Services

If the date/time stamp option (DTS) was set at SYSGEN, the WAIT and ABORT services return the date/time stamp when a task is suspended or aborted.

LOVER Service

A LOOKUP option has been added to this service to determine the presence or absence of the overlay to be loaded.

USERTRAP Service

For this service, Word 0 of the trap points to the address that follows a violating instruction in the case of a floating point overflow or transcendental exception. This is a documentation change only

Time and System Status Service (GETSYS, GETTIME, TIME)

Some of the values returned by the SYSCON format in Register 15 have been changed.

MAX IV REVISION I.0 SUMMARY

A date/time stamp option has been added to the following REX services: MESSAGE, WAIT, GETSYS, GETTIME and TIME. The date/time stamp feature has been added to the ABORT and KILL services.

An option has been added to the HEX and BTHEX Service to support conversion of a 32-bit binary value.

The GETASK and TINFORM service has been updated to provide the ability to:

- o obtain a requested FAT name
- o set or get the BAT bit in TCBRES and/or the THNA bit in TCBOOS
- o move data to and from a selected extension to the TCB.

The PROCESSOR service description has been updated to describe several special purpose REXs for MODCOMP processors.

MAX IV REVISION H.0 SUMMARY

o The following REX services have been revised to accommodate various new features of MAX IV Revision H.0:

  TASSIGN
  ACTIVATE
  KILL
  SPAWN
  ESTABLISH
  LOVER
  DUMP
  GETSYS, GETTIME, TIME

o The following new REX EXIT service subroutines are available with Revision H.0.

  I4$SCn - Condition codes are loaded according to the value of n.

  I4$X13 - Registers 1 through 13 are restored from stack and 15 entries are deleted from stack.

  I4$X14 - Registers 1 through 14 are restored from stack and 15 entries are deleted from stack.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1
INTRODUCTION

MAX IV Executive Services allow the user to "execute" privileged instructions under the supervision of the Operating System. These executive services are referred to as "REX services" or "REX calls" throughout MAX IV documentation.

Any task program (root task or overlay) can call REX services by invoking the machine instruction REX (Request Executive Service), or by using one of the many macros and interface subroutines available in higher level languages such as FORTRAN. Directly connected interrupt routines, which may be coded by the user, may not call these services.

1.1 REX SERVICE FEATURES

REX service subroutines execute privileged instructions and modify critical system data structures. An unprivileged user-coded program is not allowed such freedom. An unprivileged user-coded program can call REX services that do execute privileged instructions if such activities are supervised by the Operating System. When using REX services, there is no threat to system integrity or to other critical task programs.

REX services are reentrant subroutines located in the nucleus of the Operating System (the virtual addressing space defined by MAP 0), but they can be called from any virtual addressing space. The execution of the REX instruction causes the hardware to activate the Unimplemented Instruction Trap (interrupt level 4) and the interrupt routine is entered. A trap or any interrupt is capable of a branch across map boundaries.

Even though executing a REX instruction causes a trap at hardware interrupt level 4, this rise in priority lasts only until the service is selected. The executive service operates at the software "task level" of the invoking task, except while changing or testing certain data structures when it may be momentarily active at any of several hardware levels between level #F (Taskmaster), and level #0 (Power Failure).

1.1.1 RESIDENT OR NONRESIDENT

Some REX services are permanently resident in the nucleus, while others are loaded into dynamically allocated memory areas when invoked. In many cases, the system programmer has a choice as to whether a particular service is resident or nonresident; this choice is made at system generation time (only user installed REX services can be nonresident).

1.1.2 PASSING ARGUMENTS

Even though a REX service subroutine is located in MAP 0's addressing space, it can fetch arguments from the addressing space of the calling task, and can return such arguments into that

addressing space. Special privileged instructions are used to accomplish this movement of data from one addressing space to another. Arguments can also be passed in the registers of the calling task, which are directly accessible by the REX service subroutine; which uses the same block of general registers as the calling task program.

1.2 TWO TYPES OF REX CALLS

There are two types of REX service calls under MAX IV.

- o MAX III compatible REX calls use inline memory arguments.
- o MAX IV only REX calls pass and return arguments in general registers.

1.2.1 MAX III COMPATIBLE

The low order seven bits of the REX instruction acts as a binary number that selects the desired executive service subroutine. The MAX III compatible type of REX service call uses inline memory arguments that follow the REX instruction. This provides a convenient type of calling sequence. The REX service subroutine must skip over these arguments when returning control to the calling program. Some arguments are also passed and returned in general registers.

Use of this type of REX service call should be limited to those programs that must be written to operate under both the MAX III and MAX IV operating systems, or for MAX III programs that are to be quickly converted to run under MAX IV.

1.2.2 MAX IV ONLY

For the MAX IV only type of REX service call, arguments are passed and returned in general registers, and no inline arguments are used. While this calling scheme is not as convenient to some programs, it simplifies the nested usage of executing services within other reentrant executive services. This scheme also uses the architecture of the CLASSIC to a better advantage since the fetching of memory arguments across map boundaries adds overhead that can be avoided if memory arguments are not used.

The MAX IV only service uses the instruction:

```
REX,#32
```

for all calls and uses a general register (R8) to pass the desired service subroutine number and call options. Bits 8-15 select the particular service number to be executed, while Bits 0-7 select either binary or unitary options of the selected service. When MAX III services are installed in MAX IV, there is room for 128 services of the MAX IV only type. This number is expandable to 256 possible services if the MAX III services are not installed. MAX IV requires the MAX III compatible services if batch-processing tasks or FORTRAN-coded tasks are to be executed.

MAX IV implements most of the MAX III Executive service functions. These are listed numerically in Appendix A, Part 1. The MAX IV services are described in detail in Chapter 2. Complete descriptions of the MAX III services can be found in the MAX III reference manuals. Many service numbers are unused. Some of these are reserved for system expansion while others are available to the system programmer for implementation of custom user-coded REX services and "local" REX services.

1.3 LOCAL REX SERVICES

All undefined REX service numbers have a certain system entry point. This is also true of any other trappable unimplemented instructions (1x, 3x, or 5x op-code groups) if no software simulation routines have been provided. If these undefined instructions are executed, the entry point causes the calling task to be aborted with the reason code "REX" displayed to the system operator.

To allow use of the unassigned REX service numbers, there is an optional REX service, that may be specified at system generation time. This service (USERTRAP) permits a task to specify its own unique address within its own addressing space, where the system transfers control if the task program executes an undefined or illegal (privileged) instruction. This user-coded "trap" routine, within the calling task, can then act as a "central clearing house" for such instruction executions and may cause these instructions to be treated as "local REX services" or "local implementations of unimplemented or privileged instruction op-codes" or as "no-operations".

An advantage of this feature is that a user can test new REX services (at a local task level) before using them as resident global REX services. A user can also test simulated macro instructions before investing in hardware implementation of them. A program with privileged instructions can be checked out, to some extent, in the unprivileged mode. In programs that have a large number of subroutine calls, the local REX service capability provides for "single-word" subroutine entry calls - a saving of memory, at the expense of time.

1.4 ASSEMBLY REX CALLS

To call REX services from programs in Assembly language, call the REX services directly by supplying the appropriate arguments and option bits as hexadecimal or decimal constants and machine instructions. For example, consider a typical MAX IV type REX service call in which all arguments are specified numerically:

```
        ...
        LDI,8        #8014     SERVICE NUMBER AND OPTIONS
        LDI,14       #8005     NUMBER OF MINUTES AND TIME MODE
        LDI,15       #300      NUMBER OF TICKS
        REX,#32                SERVICE CALL
        ...                    SERVICE RETURN
```

The following example illustrates use of symbolic labels.

```
        .
DELAY   EQU          #14       REX SERVICE NUMBER FOR DELAY SERVICE
RETURN  EQU          #8000     THE "QUICK RETURN" OPTION OF SERVICE
ELAPSED EQU          #8000     THE "ELAPSED" TIME FORMAT
MAXIV   EQU          #32       THE REX SERVICE FOR ALL MAX IV CALLS
        .
```

A more symbolic form of calling the same service is possible using the labels defined above.

```
        ...
        LDI,R8        DELAY!RETURN
        LDI,R14       5!ELAPSED
        LDI,R15       #300
        REX,MAXIV
        ...
```

This type of call is much easier to use, easier to remember, and easier to maintain, but requires more writing for the programmer, particularly if each symbol must be defined in each program that uses them. The job of symbol definition has been done by MODCOMP, and is thus a step that can be avoided. To use these standard REX service symbolic labels and the register equates, use the following instructions to insert the source files into the macro-assembly program (after any macro prototype):

```
     ...
     INS*      MC,IVEQUATES
     INS*      MC,IV EQUATES
     ...
```

Chapter 2 defines the symbolic labels that may be used with each REX service.

1.5 FORTRAN REX CALLS

A programmer using a higher level language such as FORTRAN IV FR5 can call REX services directly with the INLINE and FINI statements. These statements permit the programmer to use Assembly language instructions within the FORTRAN IV program.

If program compatibility between various computers is a concern, a set of subroutines in the Executive Functions and Process I/O Library is compatible with the ISA standards for process control extensions of FORTRAN.

Additional subroutines are also included in the Utility Library. These permit all the functions of basic REX services to be used by a FORTRAN programmer. The calling sequences for these subroutines are somewhat different from the macro calls described in the previous section, but are tailored to meet the various peculiarities of the FORTRAN language. They have been made similar to the macro calls in every way possible

A FORTRAN call has the following form:

```
CALL   verb([parameter]...)
```

The same basic syntax conventions and definition techniques are used that were used to describe the macro calls.

- o The parentheses are required surrounding the parameter list.
- o Blank characters are ignored except in character strings.
- o Parameter names, expressions, and values must follow the normal rules of FORTRAN. Lower case parameters must be replaced with the names of integer variables or integer arrays, which begin with I, J, K, L, M, N, unless otherwise declared as integers.

A detailed description of FORTRAN CALLS to REX services can be found in the MAX IV Utility Library, LBM and the MAX IV Executive Functions and Process I/O Library, LBM listed in the Preface.

CHAPTER 2
MAX IV ONLY REX SERVICES

This chapter describes services that are available to programs that execute only on MAX IV Operating Systems. Refer to Chapter 3 for a list of the MAX III compatible REX services.

In this chapter, the term UFT refers to the "extended" User File Table required for the MAX IV ONLY services for I/O and not the shorter UFT used for MAX III COMPATIBLE services for I/O.

Services are listed numerically, by call number. The following table describes the organization of services and call numbers.

TYPE OF SERVICE	CALL NUMBERS
Basic Input/Output	(#00-#0C)
Task Execution Control-Self-Initiated	(#10-#14)
Execution Control - To Other Tasks	(#15-#17)
Task Scheduling	(#18-#1B)
Priority Change	(#1C)
Task Relinquish	(#1D-#1E)
Job/Task State	(#20)
Workbench	(#22)
Exclusive Use	(#23-#24)
Establish Residency	(#27-#28)
Memory Allocation	(#29-#2A)
Loader	(#2B-#2D)
Trap Control	(#2F)
Byte String Analysis	(#34-#35)
Conversion	(#37-#3C)
Event Logging	(#3D)
Roll	(#3E)
Memory Dump	(#3F)

Time and System Status (#40)

Intertask Communications (#42)

Task Status (#43)

Task Option and Variable Control (#50-#56)

Modification of Access Rights (#57-#5C)

Standard REX service symbolic labels are used in the examples appearing in this manual. The equate statements (EQU) defining these labels are contained in a source file named IVEQUATES. Additional symbolic labels used in the examples are contained in the IVJEQU source file.

The following statements must be used to include these equate files in an Assembly program source.

```
INS*      MC,IVEQUATES
INS*      MC,IVJEQU
```

The first six characters of the symbolic labels are unique. The user need specify only the first six characters of these symbolic labels.

In this chapter, descriptions of REX services are presented in the following format:

CALL NAME

SERVICE SUMMARY

Call Name: Call Number:

Option Name: Option Value:

SERVICE PERFORMED

CALLING PARAMETERS

CONDITIONS CODES UPON RETURN

PROGRAMMING CONSIDERATIONS

USAGE EXAMPLE

READ ONE PHYSICAL RECORD FROM I/O DEVICE

Call Name: READ   Call Number: $0 (REX,MAXIV

Option Name: QUICK   Option Value: $8000

SERVICE PERFORMED

This service reads a single physical record from the I/O device assigned to the specified logical file. The size of the physical record is determined by the device record size specified at system generation time, or by the user's buffer size rounded to the next natural device boundary if the device size is unlimited. The largest transfer, without using data chaining, is 16K words (32K bytes).

CALLING PARAMETERS

R2: UFT address (refer to Appendix B

R0: Service option and call number:
 Bit 0: Return mode bit
 =0 If Wait Mode (return when operation completed).
 =1 If Quick Return Mode (return when operation queued). Option name = QUICK.
 Bits 1-7: =0
 Bits 8-15: Service call number = $0 and call name = READ.

[R12:] First 3 characters (CAN-code) of task name (if not calling task,
[R13.] Second 3 characters (CAN-code) of task name (if not calling task
[R14. Input Buffer Address or Chain List Pointer (may be in UFT)
R15.] Transfer Count (bytes) or Chain Control Word (may be in UFT)

CONDITION CODES UPON RETURN

NZOC Condition

*1000 - Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

The feature allowing the calling task to use the logical file assignments of another task (as specified by R12/13) is usable only by privileged tasks when referencing another task's File Assign Table (FAT). These registers are ignored otherwise. This feature is selected by setting a bit in the Extended Option Word of the UFT.

The input buffer for reading the requested record (or the chain that describes a scatter read "buffer") is assumed to be in the operand addressing space of the calling program. A privileged task can specify other map image tables, and the data chain can be in MAP 0 while referencing virtual addresses to be translated by the hardware in the specified table.

The data chaining feature is device dependent, and currently operable only for devices that use a Direct Memory Processor (DMP). Devices using data interrupts do not simulate this feature unless noted in their handler descriptions. Only privileged programs may use data chaining.

The buffer address and byte count (or chain address) can be specified in an optional extension of the UFT which causes R14/15 to be ignored. The option to use R14/15 is specified by a bit in the Extended Option Word of the UFT.

The Quick Return Mode may be specified as an option in R8 or as an option in Word 6 (Extended Option Word) of the UFT. This bit set in either place yields Quick Return Mode, so it need not be set in both places.

USAGE EXAMPLE

Read a card into a buffer in my operand space.

```
*                             BIT 1 OF WORD 6 OF UFTADR IS SET.
     ...
     LDI,R2     UFTADR        UFT ADDRESS OF FILE ASSIGNED TO CR.
     LDI,R14    BUFF          BUFFER ADDRESS.
     LDI,R15    80            80 BYTES
     LDI,R8     READ          USE READ SERVICE IN WAIT MODE.
     REX,MAXIV                REQUEST THE INDICATED SERVICE.
*    .                        RETURN HERE WHEN SERVICE COMPLETED.
```

WRITE ONE PHYSICAL RECORD TO I/O DEVICE

Call Name: WRITE   Call Number: #1 (REX,MAXIV)

Option Name: QUICK   Option Value: #8000

SERVICE PERFORMED

This service writes a single physical record to an I/O device assigned to the specified logical file. The size of the physical record is determined by the system's definition of the device record size, or if unlimited in size, it is determined by the user's buffer size rounded up to the next natural device record boundary. The largest transfer, without using data chaining, is 16K words (32K bytes).

CALLING PARAMETERS

R2: UFT address (refer to Appendix B)

R8: Service option and call number:
Bit 0: Return mode bit:
=0 If Wait Mode (return when operation completed).
=1 If Quick Return Mode (return when operation queued). Option name = QUICK.
Bits 1-7: =0
Bits 8-15: Service call number = #1 and call name = WRITE.

R12] First 3 characters (CAN-code) of task name (if not calling task)

R13] Second 3 characters (CAN code) of task name (if not calling task)

R14] Input Buffer Address or Chain List Pointer (may be in UFT)

R15] Transfer Count (bytes) or Chain Control Word may be in UFT

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

The feature allowing the calling task to use the logical file assignments of another task as specified by R12/13, is usable only by privileged tasks when referencing another task's File Assign Table (FAT). These registers are ignored otherwise. This feature is selected by setting a bit in the Extended Option Word of the UFT.

The output buffer for writing the requested record or the chain that describes a gather write "buffer" is assumed to be in the operand addressing space of the calling program. A privileged task can specify other map image tables, and the data chain can be in MAP 0 while referencing virtual addresses to be translated by the hardware in the specified table.

The data chaining feature is device dependent, and currently operable only for devices that use a Direct Memory Processor (DMP). Devices using data interrupts do not simulate this feature unless noted in their handler descriptions. Only privileged programs may use data chaining.

The buffer address and byte count (or chain address) can be specified in an optional extension of the UFT which causes R14/15 to be ignored. The option to use R14/15 is specified by a bit in the Extended Option Word of the UFT.

The Quick Return Mode may be specified as an option in R8 or as an option in Word 6 (Extended Option Word) of the UFT. This bit set in either place yields Quick Return Mode, so it need not be set in both places.

USAGE EXAMPLE

Write a line to the printer.

```
        ...
        LDI,R2      UFT       UFT ADDRESS OF FILE ASSIGNED TO LP.
        LDI,R14     BUFF      BUFFER ADDRESS.
        LDI,R15     134       134 BYTES.
        LDI,R8      WRITE     USE WRITE SERVICE IN WAIT MODE.
        REX,MAXIV             REQUEST THE SERVICE.
*        ...                  RETURN HERE WHEN OPERATION COMPLETED.
```

REWIND I/O DEVICE

Call Name: REWIND   Call Number: #2 (REX,MAXIV)

Option Name: QUICK   Option Value: #8000

SERVICE PERFORMED

This service positions a device assigned to a specified logical file to the Beginning of Medium (BOM) position if such a position is defined for the device, either physically or logically

CALLING PARAMETERS

R2: UFT address (refer to Appendix B).

R8: Service Option and call number:
Bit 0: Return mode bit:
=0 if Wait Mode (Return when operation completed).
=1 if Quick Return Mode (Return when operation queued). Option name = QUICK.
Bit 1-7: =0
Bits 8-15: Service call number = #2 and call name = REWIND.

[R12:] First three characters of CAN-code task name (if not calling task).
[R13: Second three characters of CAN-code task name (if not calling task).

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	— Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

R12/13 are used only by privileged tasks if they select another task's logical files. These registers are ignored otherwise.

USAGE EXAMPLE

Rewind a magnetic tape.

```
        ...
        LDI,R2      UFTADR      UFT ADDRESS OF FILE ASSIGNED TO MT1.
        LDI,R8      REWIND      SPECIFY REWIND SERVICE IN WAIT MODE.
        REX,MAXIV               REQUEST THE SERVICE.
*       ...                     RETURN HERE WHEN OPERATION COMPLETE.
```

BACKSPACE A FILE ON I/O DEVICE

Call Name: BKFILE   Call Number: #3 (REX,MAXIV)

Option Name: QUICK   Option Value: #8000

SERVICE PERFORMED

If the device is not already at its Beginning of Medium (BOM), the service backspaces over physical records until the file mark (EOF) physical record is found, or until the Beginning of Medium (BOM) position is reached. The next record accessed with a forward device operation will be the EOF record except when the operation results in positioning the device at Beginning of Medium (BOM).

CALLING PARAMETERS

R2: UFT address (refer to Appendix B).

R8: Service Option and call number:
Bit 0: Return mode bit:
=0 if Wait Mode (Return when operation completed).
=1 if Quick Return Mode (Return when operation queued). Option name = QUICK.
Bits 1-7: =0
Bits 8-15: Service call number = #3 and call name = BKFILE.

[R12:] First three characters of CAN-code task name (if not calling task).
[R13: Second three characters of CAN-code task name (if not calling task).

CONDITION CODES UPON RETURN

NZOC Condition

'1000 - Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

On completion of the operation, the device will be positioned on the EOF record (that is, a READ, after a BKFILE yields EOF status). This is not true if the device reaches Beginning of Medium.

R12/13 are used only for privileged tasks to use another task's File Assign Table (FAT). These registers are ignored otherwise.

USAGE EXAMPLE

Backup the mag tape to the EOF record before the start of my current file.

```
          LDI,R2     UFT      UFT ADDRESS OF FILE ASSIGNED TO MT1.
          LDI,R8     BKFILE   SPECIFY BKFILE SERVICE IN WAIT MODE.
          REX,MAXIV           REQUEST THE SERVICE.
*         ...                 RETURN HERE AFTER OPERATION COMPLETED.
```

BACKSPACE A RECORD ON I/O DEVICE

Call Name: BKRECORD   Call Number: $4 (REX,MAXIV)

Option Name: QUICK   Option Value: $8000

SERVICE PERFORMED

This service positions a device at the beginning of the previous record (if not already at Beginning of Medium).

CALLING PARAMETERS

```
        R2:    UFT address (refer to Appendix B).
        R8:    Service Option and call number:
               Bit 0:     Return mode bit.
                          =0 if Wait Mode (Return when operation
                             completed).
                          =1 if Quick Return Mode (Return when
                             operation queued).  Option name =
                             QUICK.
               Bits 1-7:  =0
               Bits 8-15: Service call number = $4 and call name =
                            BKRECORD.

      [R12:    First three characters of CAN-code task name (if not
                 calling task).
      [R13:    Second three characters of CAN-code task name (if
                 not calling task).
```

CONDITION CODES UPON RETURN

```
      NZOC     Condition

      *1000 -  Normal return (all status bits are in UFT's status
               word since completion may be asynchronous with respect
               to the call).
```

PROGRAMMING CONSIDERATIONS

R12,13 are used only by privileged tasks if they select another task's logical files. These registers are ignored otherwise. If the backspace record service positions the device at an End-of-File record, then the EOF bit is set in the UFT.

USAGE EXAMPLE

Backup the magnetic tape to the previous record.

```
      LDI,R2      UFTADR      UFT ADDRESS OF FILE ASSIGNED TO MT1.
      LDI,R8      BKRECORD    SPECIFY BKRECORD SERVICE IN WAIT MODE.
      REX,MAXIV               REQUEST THE SERVICE.
*      ...                    RETURN HERE AFTER OPERATION COMPLETE.
```

ADVANCE A RECORD ON I/O DEVICE

Call Name: AVRECORD   Call Number: $5 (REX,MAXIV)

Option Name: QUICK   Option Value: $8000

SERVICE PERFORMED

This service positions a device to the beginning of the next record (skip one record).

CALLING PARAMETERS

```
R2:     UFT address (refer to Appendix B).
R8:     Service Option and call number:
        Bit 0:      Return mode bit:
                    =0 if Wait Mode (Return when operation
                       completed).
                    =1 if Quick Return Mode (Return when
                       operation queued).  Option name =
                       QUICK.
        Bits 1-7:   =0
        Bits 8-15:  Service call number = $5 and call name =
                      AVRECORD.

[R12:]  First three characters of CAN-code task name (if not
          calling task).
[R13:]  Second three characters of CAN-code task name (if
          not calling task).
```

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

R12/13 are used only by privileged tasks if they select another task's logical files. These registers are ignored otherwise. If the advance record service positions the device at an End-of-File record, then the EOF bit is set in the UFT.

USAGE EXAMPLE

Skip a record on magnetic tape

```
        LDI,R2      UFTADR    UFT ADDRESS OF FILE ASSIGNED TO MT1.
        LDI,R8      AVRECORD  SPECIFY THE AVRECORD SERVICE IN THE WAIT MODE.
        REX,MAXIV             REQUEST THE SERVICE.
*          ...                RETURN HERE AFTER OPERATION COMPLETED.
```

AVFILE

ADVANCE A FILE ON I/O DEVICE

Call Name: AVFILE　　Call Number: $6 (REX,MAXIV)

Option Name: QUICK　　Option Value: $8000

SERVICE PERFORMED

This service positions a device to beyond the next End of File record.

CALLING PARAMETERS

```
R2:     UFT address (refer to Appendix B).
R8:     Service Option and call number.
        Bit 0:      Return mode bit:
                    =0 if Wait Mode (Return when operation
                       completed).
                    =1 if Quick Return Mode (Return when
                       operation queued).  Option name =
                       QUICK
        Bits 1-7:   =0
        Bits 8-15:  Service call number = $6 and call name =
                      AVFILE.

[R12!]  First three characters of CAN-code task name (if not
          calling task).
[R13!]  Second three characters of CAN-code task name (if
          not calling task).
```

CONDITION CODES UPON RETURN

NZOC　Condition

'1000 - Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call)

PROGRAMMING CONSIDERATIONS

R12/13 are used only by privileged tasks if they select another task's logical files. These registers are ignored otherwise. If the End-of-Medium is reached before the End-of-File record is encountered, then the EOM bit is set.

USAGE EXAMPLE

Go to next magnetic tape file.

```
        LDI,R2      UFTADR     UFT ADDRESS OF FILE ASSIGNED TO MT1.
        LDI,R8      AVFILE     SPECIFY AVFILE SERVICE IN WAIT MODE.
        REX,MAXIV              REQUEST THE SERVICE.
*       ...                    RETURN WHEN SERVICE COMPLETED.
```

WRITE AN END-OF-FILE MARK ON THE I/O DEVICE

Call Name: WEOF   Call Number: #7 (REX,MAXIV)

Option Name: QUICK   Option Value: #8000

SERVICE PERFORMED

This service writes an EOF record on the I/O device assigned to the specified logical file.

CALLING PARAMETERS

R2: UFT address (refer to Appendix B).
R8: Service Option and call number:
  Bit Ø: Return mode bit:
    =Ø if Wait Mode (Return when operation completed).
    =1 if Quick Return Mode (Return when operation queued). Option name = QUICK.
  Bits 1-7: =Ø
  Bits 8 15: Service call number = #7 and call name = WEOF.

[R12:] First three characters of CAN-code task name (if not calling task).
[R13: Second three characters of CAN-code task name (if not calling task).

CONDITION CODES UPON RETURN

NZOC	Condition
*1000 -	Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

R12/13 are used only by privileged tasks if they select another task's logical files. These registers are ignored otherwise. The End-of-File record written on a readable sequential device will be recognized by standard MODCOMP processors.

USAGE EXAMPLE

Write an EOF record on the magnetic tape.

```
        LDI,R2      UFTADR     UFT ADDRESS (PR  OF FILE ASSIGNED TO MT1.
        LDI,R8      WEOF       SPECIFY THE WEOF SERVICE IN WAIT MODE.
        REX,MAXIV              REQUEST THE SERVICE.
*       ...                    RETURN HERE AFTER OPERATION COMPLETED.
```

NORMALIZE AN I/O DEVICE

Call Name: HOME　　　　Call Number: #8 (PEX,MAXIV)

Option Name: QUICK　　　　Option Value: #8000

SERVICE PERFORMED

This service normalizes most devices (and performs no operation for others).

CALLING PARAMETERS

R2: UFT address (refer to Appendix B).

R8: Service Option and call number:
- Bit 0: Return mode bit:
  - =0 if Wait Mode (Return when operation completed).
  - =1 if Quick Return Mode (Return when operation queued). Option name = QUICK.
- Bits 1-7: =0
- Bits 8-15: Service call number = #8 and call name = HOME.

[R12:] First three characters of CAN-code task name (if not calling task).

[R13:] Second three characters of CAN-code task name (if not calling task).

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

R12/13 are used only by privileged tasks if they select another task's logical files. These registers are ignored otherwise.

This service should be called when a program first uses a device that might be in an unknown state. The HOME returns the device to a normalized state. The response to this service is device/handler dependent. Refer to the description of the Handler in one of the MAX IV handler manuals.

USAGE EXAMPLE

Clear the hardware mag-tape device buffer.

```
        LDI,R2      UFTADR   UFT ADDRESS OF FILE ASSIGNED TO MT.
        LDI,R8      HOME     SPECIFY THE HOME SERVICE IN WAIT MODE.
        REX,MAXIV            REQUEST THE SERVICE.
*       ...                  RETURN HERE AFTER SERVICE COMPLETED.
```

TERMINATE AN I/O DEVICE

Call Name: TERMINATE   Call Number: #9 (REX,MAXIV,

Option Name: ALL   Option Value: #8000

SERVICE PERFORMED

This service terminates operations queued to a device by the calling task.

CALLING PARAMETERS

R2: UFT address (Refer to Appendix B).

R8: Service option and call number:
Bit 0: Class of operation:
=0 termination of operations queued by calling task.
=1 termination of all operations queued for this device regardless of calling task. Option name = ALL.
Bits 1-7: =0
Bits 8-15: Service call number = #9 and call name = TERMINATE.

[R12.] First 3 characters of CAN-code task name (if not calling task).

R13.] Second 3 characters of CAN-code task name (if not calling task).

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

Only a privileged task can use the ALL option. Bit 0 of the Extended Option Word in the UFT is inclusively OR'ed with option Bit 0 of Register 8. That is, if either bit is set the ALL option is selected

R12/13 is used when a privileged task uses another task's File Assign Table (FAT). These registers are ignored otherwise.

USAGE EXAMPLE

Terminate the magnetic tape device.

```
       LDI,R2      UFTADR        UFT ADDRESS OF FILE ASSIGNED TO MT1.
       LDI,R8      TERMINATE     SPECIFY THE TERMINATE SERVICE.
       REX,MAXIV                 REQUEST THE SERVICE.
*      . .                       RETURN HERE AFTER SERVICE COMPLETED.
```

ASSIGN

LINK A LOGICAL FILE TO AN I/O DEVICE OR ANOTHER LOGICAL FILE

Call Name:	ASSIGN	Call Number:	$A (REX,MAXIV)
Option Names:	DEFAULT	Option Values:	$8000
	MOVE		$4000
	NOWBOT		$2000
	DEVPRE		$1000

SERVICE PERFORMED

This service associates a referenced "entity" with the specified logical file in the File Assign Table (FAT) of the calling task. The specified logical file is searched for in the task's FAT and if not found, a vacant entry (if available) is used. The service assumes the entity to be assigned to the specified logical file is a reference to another logical file already assigned in the task's FAT. If no such logical file exists in the task's FAT, the service assumes the entity is a reference to a logical file already assigned in the system's global FAT. If the logical file does not exist in the system's global FAT, the service assumes the entity is a reference to a system device. This search precedence of a task's FAT, global FAT, and system device list can be changed with the DEVPRE option. If that option is used, the search precedence is the system device list, the task's, FAT, and the global FAT.

If the referenced entity is a system device and the DEFAULT option is NOT specified, the current assignment of the specified logical file becomes the referenced device. The File Position Index (FPI) in the FAT and the User File Table (UFT) for the specified logical file is reset to zero. If the DEFAULT option is specified, the default assignment of the specified logical file becomes the referenced device. The FPI in the FAT and UFT for the specified logical file is unaffected.

If the referenced entity has the same name as the specified logical file (i.e., references itself), and the DEVPRE option was not specified or if it was specified and the referenced entity is not in the system device list, and the DEFAULT option was NOT specified, the current assignment of the specified logical file becomes the referenced logical file. The FPI found in the FAT of the referenced logical file is copied into the FPI of the FAT and UFT for the specified logical file. If the DEFAULT option is specified, the default assignment of the specified logical file becomes the referenced logical file. The FPI in the FAT and UFT for the specified logical file is unaffected.

If the referenced entity is not a system device and is the same name as that of the specified logical file (i.e., references itself) and the DEFAULT option is NOT specified, a check is made to see if the specified logical file has a default assignment. If a default assignment exists, the current assignment of the specified logical file becomes the default assignment. If no default assignment exists, the current assignment of the specified logical

file is "unassigned" (cleared). Also, if the specified logical file was not cataloged as a "permanent" entry the file will return to the vacant state. In either case the FPI in the FAT and UFT for the specified logical file is reset to zero. If the DEFAULT option is specified, the default assignment of the specified logical file is unassigned (whether or not a default assignment exists). The FPI in the FAT and UFT for the specified logical file is unaffected.

If the referenced entity is zero, the request is considered to be a "close" of the specified logical file. With or without the DEFAULT option, a close results in the unassigning of the current and default assignments of the specified logical file. If the specified logical file was not cataloged as a permanent entry the file returns to the vacant state. If the DEFAULT option is NOT specified, the FPI in the FAT and UFT for the specified logical file is reset to zero in addition to closing the specified logical file.

If the MOVE option is selected (Bit 1 of Register 8 is set), the current assignment of the logical file name in Register 15 is made to look like the current assignment of the logical file name in the UFT pointed to by Register 2. To accomplish this, the FAT corresponding to the logical file name in the UFT pointed to by Register 2 is copied to the FAT of the logical file name in Register 15. This copy duplicates the current assignment pointer and the FPI. If no FAT entry exists for the logical file name in Register 15, a vacant FAT entry is allocated. The DEFAULT option is not supported for this feature.

CALLING PARAMETERS

R2 UFT address of file to be assigned or opened (refer to Appendix B).

R8: Service option and call number

Bit 0: =0 Change current assignment
=1 Change default assignment. Option name = DEFAULT.

Bit 1: =0 Do not include the FPI
=1 Move the current assignment including the FPI associated with R2 to R15. Option name = MOVE.

Bit 2: =0 Issue REX WEOF when assignment is made away from target lfn or device that is a printer (non-terminal) symbiont device.
=1 DO NOT issue a REX WEOF when assignment is made away from target lfn or device that is a printer (non-terminal symbiont device. Option name = NOWEOF.

Bit 3: =0 Search for the referenced entity using the following precedence: task's FAT, global FAT, system device list.
=1 Search for the referenced entity using the following precedence: system device list, task's FAT, global FAT. Option name = DEVPRE

Bits 4-7: =0
Bits 8 15: Service call number = #A and call name = ASSIGN.

[R12:] First 3 characters of CAN-code task name (if not calling task).

[R13:] Second 3 characters of CAN-code task name (if not calling task).

R15: 3-character CAN-code name of the referenced entity, that is, the device or other logical file to be assigned (or zero if to be unassigned).

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	- Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

R12/13 is used only for privileged tasks to use another task's FAT. These registers are ignored otherwise. Bit OTF in the Extended Option Word of the UFT must be set to use this option.

The service checks for and does not allow a specified logical file of zero.

The service checks for and does not allow an assignment that would result in a file-to-file loop through either the current or default assignment of the specified logical file.

The service checks for and does not allow the current or default assignment of the specified logical file to be unassigned if there are any other logical files in the same FAT whose current or default assignments point to the specified logical file. (The same applies to vacating a specified logical file since this in effect unassigns the current and default assignments.) The "other" logical files whose current or default assignments point to the specified logical file must have their current and/or default assignments changed before the specified logical file can be vacated or have its current or default assignments unassigned.

USAGE EXAMPLES

Assign my file (SI) to the card reader (CR).

```
        LDI,R2      UFTADR        UFT ADDRESS WITH FILE NAME (SI)
        LDI,R 5     @CR           SPECIFY DEVICE NAME CARD READER.
        LDI,R8      ASSIGN        SPECIFY CURRENT FILE ASSIGNMENT.
        REX,MAXIV                 REQUEST THE SERVICE.
*       ...                       RETURN HERE WHEN ASSIGNMENT COMPLETE
```

Set the default device for my file SC to magnetic tape drive 1 (MT1).

```
        LDI,R2      UFTADR       UFT ADDRESS WITH FILE NAME (SO).
        LDI,R15     @MT1         SPECIFY DEVICE NAME MAGNETIC TAPE 1.
        LDI,R8      ASSIGN DEFAULT
*                                SPECIFY DEFAULT ASSIGNMENT.
        REX,MAXIV                REQUEST THE SERVICE.
*       ...                      RETURN HERE.
```

Make the default assignment for my file (SO), the current assignment.

```
        LDI,R2      UFTADR       UFT ADDRESS WITH FILE NAME (SO).
        LDI,R15     @SO          ASSIGN FILE TO ITSELF.
        LDI,R8      ASSIGN       MODIFY CURRENT ASSIGNMENT.
        REX,MAXIV                REQUEST THE SERVICE.
*       ...                      RETURN HERE.
```

Remove file XXX from the File Assign Table.

```
        LDI,R2     UFTADR        UFT ADDRESS WITH XXX AS FILE NAME.
        LDI,R8     ASSIGN.DEFAULT
*                                CHANGE DEFAULT ASSIGNMENT.
        ZRR,R15                  SET TO ZERO.
        REX,MAXIV                REQUEST THE SERVICE.
        LDI,R8     ASSIGN        CHANGE CURRENT ASSIGNMENT.
        LDI,R15    @XXX          ASSIGN THE FILE TO ITSELF.
        REX,MAXIV                REQUEST SERVICE AGAIN, DELETE ENTRY.
*       ...                      XXX IS DELETED FROM THE FAT.
```

Move current assignment (including the FPI) of file xxx to file SI

```
        LDI,R2     UFTADR
        LDI,R8     ASSIGN.MOVE
        LDI,R15    @SI
        REX,MAXIV
```

NOTE: The MOVE option can be used to remember positions within a file.

```
UFTADR DFC        0,@XXX,#A000, ...
```

TEST ASSIGNMENT OF A LOGICAL FILE TO A DEVICE

Call Name:	TASSIGN	Call Number:	#B (REX,MAXIV
Option Names:	DISC	Option Values:	#8000
	TOEFAULT		#4000
	INMEM		#2000
	FMENAME		#4000

SERVICE PERFORMED

The I/O device assigned to either the current or default logical file is checked by resolving intermediate logical file assignments Information about the physical and logical characteristics are returned to the caller.

CALLING PARAMETERS

(R1:) Address of snap block if information is to be returned to memory. (Refer to the section entitled Memory Information Returned further in this call description.)

R2: UFT address (refer to Appendix B).

R8: Service options and call number specified as:
Bit 0: Type of information:
=0 Get standard information.
=1 Get disc information. Option name = DISC.
Bit 1: Assignment Selection:
=0 Use current assignment.
=1 Use default assignment. Option name = TOEFAULT
Bit 2: Where to return information:
=0 In registers.
=1 In snap block. Option name = INMEM.
Bits 3-7: Unitary Information Selection Field.
- o Standard device information is selected by:
  - Bit 3 = Return device's status/record size information
  - Bit 4 = Return exclusive use information
  - Bit 5 = Return current File Position Index
  - Bit 7 = Use extended option register (R9).
- o Disc device information is selected by:
  - Bit 3 = Return pack name
  - Bit 4 = Return disc environment information
  - Bit 5 = Return disc coordinates of device
  - Bit 6 = Return physical transport information
  - Bit 7 = Use extended option register (R9).

Bits 8-15: Service call number =#B and call name = TASSIGN.

R9: Extended option register, has meaning only if Bit 7 of R8 is set.
Bit 0: Reserved for MAXNET ,see MAXNET documentation.

Bit 1: The name of the File Manager device is returned instead of a -2 value (#FFFE) when
- o Bit 0 of R8 is reset (standard information) and
- o Bit 3 of R8 is set (return device status/record size) and
- o the logical file name specified is assigned to a File Manager FAT.

Bit 2: The LDT address of the UFT logical file name (UFNAM) is returned in R3 when:
- o Bit 7 of R8 is set (use extended option register - R9)

Bit 3: DCS information is returned when:
- o Bit 0 of R8 is reset (standard information) and
- o Bit 7 of R8 is set (use extended option register - R9)

[R12:] First 3 characters (CAN-code) of task name (if not calling task).
[R13:] Second 3 characters (CAN-code) of task name (if not calling task).

REGISTER INFORMATION RETURNED

o Standard device information:

If Selection Bit 3 is set to one:

R8: File status/device size information:
- Bit 0: Logical file is global.
- Bit 1: Assignment of file is to another logical file.
- Bit 2: Assigned device is offline.
- Bit 3: Logical file is permanent
- Bit 4: Assigned device is imaginary (attached to symbiont).
- Bit 5: Assigned device is under permanent exclusive use
- Bit 6: MAXNET device.
- Bit 7: --unassigned--
- Bits 8-15: Assigned device maximum physical record size in words (=0 if unlimited).

R9: Assigned device name in CAN-code (3 characters) or a minus two (#FFFE) indicating a File Manager file.

If the FMFNAME option is selected (Bit 1 of R9 set), the File Manager device name is returned instead of a minus two for File Manager files.

R10: Assigned device handler options/device class:
- Bits 0-7: Device handler options (see handler descriptions).
- Bit 8: Immediate class device assigned.
- Bit 9: Plotter class device assigned.
- Bit 10: Printer class device assigned.
- Bit 11: Card class device assigned.
- Bit 12: Paper-tape class device assigned.

Bit 13: Terminal (interactive) device class assigned
Bit 14: Magnetic-tape class device assigned.
Bit 15: Random-access (disc) class device assigned.

If Selection Bit 7 is set to one and Bit 3 of extended option register (R9) is set to one:

R11: DCS information:

Bit 0: Set if DCS device
Bits 1-15: Reserved

If Selection Bit 4 is set to one:

R12: Name of task using device exclusively (first 3 characters of CAN-code).
R13: Name of task using device exclusively (last 3 characters of CAN-code).

If Selection Bit 5 is set to one:
R14: File/Device Position Index (Most-significant half).
R15: File/Device Position Index (Least-significant half).

o Disc information (when the DISC option is selected):

If Selection Bit 3 is set to one:
R7: Transport (pack) name in CAN-code (3 characters).

If Selection Bit 4 is set to one:
R8: Number of words per-sector
R9: Number of sectors-per-track,head/revolution)
R10: Total number of tracks on pack
R11: Two bytes as follows:
Bits 0-7: Number of tracks(heads)-per-cylinder
Bits 8-15: Transport status where:
Bit 8: Volume is mounted
Bit 9. Pack Exclusive Use allowed
Bit 11: Device is an AFD and uses DMI
Bit 12: Device is an AFD (implies Bit 14 is also set)
Bit 13: Cylinder boundary overflow not supported
Bit 14: Floppy Disc
Bit 15: Fixed-Head Disc

If Selection Bit 5 is set to one:
R12: Starting sector of disc file/device (most-significant half).
R13: Starting sector of disc file/device (least-significant half).
R14: Total sectors in file (most-significant half).
R15: Total sectors in file (least-significant half).

If Selection Bit 6 is set to one:
R9 through R10 above reflect physical transport characteristics instead of logical transport characteristics

MEMORY INFORMATION RETURNED (SNAP BLOCK)

When the INMEM option is selected on the REX call, a memory snap-block is chosen for the return of information of this service. The registers described above are not changed. The sequential memory cells of the snap block are modified with the same information described for register return, and in the order of requested information. For example, if "Standard" information is returned and selection Bit 5 is the only bit set, then only Words 0 and 1 in memory are changed with the File/Device Position Index.

CONDITION CODES UPON RETURN

NZOC   Condition

'1000 - Information returned.
'0000 - Disc information requested for a non-disc device.

PROGRAMMING CONSIDERATIONS

The feature allowing the calling task to use the logical file assignments of another task (as specified by R12/13) can only be used by privileged tasks when referencing another task's File Assign Table (FAT). These registers are ignored otherwise. This feature is selected by setting a bit in the Extended Option Word of the UFT.

I/O Error conditions are reported in the status word of the UFT.

USAGE EXAMPLES

Get current File Position Index of my file.

```
        LDI,R2      UFTADR          UFT ADDRESS.
        LDI,R8      TASSIGN!#400    GET CURRENT INFORMATION IN REGISTERS
        REX,MAXIV                   REQUEST THE SERVICE
*       ...                         RETURN HERE WITH FPI IN R14/R15.
```

Place all information about my disc in Snap Block

```
        LDI,R1      SBLOK           ADDRESS OF SNAP BLOCK IN OPERAND SPACE.
        LDI,R2      UFT             UFT ADDRESS.
        LDI,R8      TASSIGN.DISC+INMEM!#1C20
*                                   GET CURRENT DISC INFORMATION IN MEMORY.
        REX,MAXIV                   REQUEST THE SERVICE.
*       ...                         RETURN HERE.
```

IOWAIT

WAIT FOR COMPLETION OF I/O OPERATIONS

Call Name:	IOWAIT	Call Number:	#C (REX,MAXIV)
Option Names:	ALL UFTLIST MAP0 NOUFT	Option Values:	#8000 #4000 #2000 #1000

SERVICE PERFORMED

This service puts the calling task in hold until one or more I/O operations it has queued have completed. The task is momentarily resumed each time one of the specified operations is completed.

CALLING PARAMETERS

R2 UFT address (if a single UFT is specified) or the address of a list of UFT addresses (if multiple UFTs are specified).

R0: Service option and call number
- Bit 0: Conditions of task suspension
  - =0 Until any operation completes.
  - =1 Until all operations complete. Option name = ALL.
- Bit 1: UFT Information
  - =0 Single UFT Address in R2.
  - =1 Address of UFT list in R2. Option name = UFTLIST.
- Bit 2. List Location:
  - =0 List in same map as UFTs.
  - =1 List in MAP 0. Option name = MAP0.
- Bit 3: Augment to Suspend Condition
  - =0 Suspend as specified above.
  - =1 Suspend until all currently queued I/O operations of the calling task are completed (no UFTs supplied). Option name = NOUFT.
- Bits 4-7 =0
- Bits 8-15: Service call number = #C and call name = IOWAIT.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

If the specified criteria are satisfied prior to service entry, an immediate return occurs, and the task is not suspended

If MAP 0 is the operand map in use at the time the REX service is called, the UFT(s) is assumed to be in MAP 0 (otherwise the UFT is assumed to be in the Task's operand map).

Similarly, if Bit 2 of R8 is reset and MAP 0 is the operand map in use at the time the REX service is called, the list of UFT s) will be fetched from MAP 0 (otherwise fetched from the Task's operand map).

USAGE EXAMPLES

Suspend me until my I/O operation completes.

```
        LDI,R2    UFTADR     UFT ADDRESS OF OPERATION QUEUED.
        LDI,R8    IOWAIT     SUSPEND TASK UNTIL OPERATION COMPLETED.
        REX,MAXIV            REQUEST THE SERVICE.
*        .                   RETURN WHEN THIS UFT IS NO LONGER BUSY.
```

Suspend me until several specified operations have completed.

```
        LDI,R2     ULIST     ADDRESS OF LIST OF UFTS.
        LDI,R8     IOWAIT!ALL!UFTLIST
*                            SUSPEND TASK UNTIL ALL ARE COMPLETE
        REX,MAXIV            REQUEST THE SERVICE
*       ...                  RETURN HERE WHEN OPERATIONS COMPLETE.


ULIST   DFC        UFT1      LIST OF UFT ADDRESSES.
        DFC        UFT2
        DFC        UFT3
        DFC        UFT4
        DFC        0         ZERO TERMINATES LIST.
```

Suspend me until all my I/O completes. I do not know where my UFTs are.

```
        LDI,R8     IOWAIT.NOUFT
*                            WAIT/ALL OPERATIONS QUEUED BY TASK.
        REX,MAXIV            WAIT HERE.
*       ...                  RETURN WHEN ALL I/O IS COMPLETE.
```

MESSAGE

PRINT MESSAGE TO OPERATOR

Call Name:	MESSAGE	Call Number:	#10 (REX,MAXIV)
Option Name:	HOLD	Option Value:	#8000
	FNRG		#4000
	RESOURCE		#2000
	STAMP		#1000

SERVICE PERFORMED

This service permits the calling task to print messages to the "system" operator. These messages are directed to the CO file, or the file name contained in R14 (if optioned), and are printed in the Wait Mode. The task's service buffer is used to format the message and to add the calling task/overlay name to the message. If the HOLD option is set (=1, a "<HOLD>" message is also added. No intermediate buffering and no task name or hold message is added if this service is called by the operator communication task (OC). If the calling task has a local CO file, the destination of the message may be directed to a remote terminal or spooled printer unique to the calling task. If the resource option bit is set =1 the task voluntarily gives up its maps and register block while it is inactive. These resources are returned to the available pool for reallocation to other tasks.

CALLING PARAMETERS

```
R8:   Service option and call number:
      Bit 0:   Task disposition after message is printed:
               =0 Task is to continue execution
               =1 Task is to enter HOLD state (if not
                  remote) or task is to process a local
                  directive on "OC" file (if a remote task).
                  Option name = HOLD
      Bit 1:   Output file specification:
               =0 if the message is to be output to the CO
                  file
               =1 if the message is to be output to the
                  file specified in Register 14.  Option
                  name = FNRG
      Bit 2:   Resource Specification:
               =0 Task is to retain its resources
               =1 Task voluntarily returns its resources
                  to the system while inactive.
                  Option name = RESOURCE
      Bit 3:      Date/Time stamp.
               =0 Do not print date/time stamp with message.
               =1 Print date/time stamp with message.
                  Option name = STAMP.
      Bits 4-7:  =0
```

Bits 8 15: Service call number = $10 and call name = MESSAGE.

[R14:] File name in CAN-code to which the message is output if Bit 1 of Register 8 is set to 1.

R15: Address of the first word of message character string in memory, and must contain carriage control byte as first character.

CONDITION CODES UPON RETURN

NZOC	Condition
*1000	- Service performed - no error conditions defined.

PROGRAMMING CONSIDERATIONS

The data format of the message follows the rules of standard ASCII data format for printing devices. The first byte (left byte) in the first word of the message is not printed and is interpreted as a carriage positioning function. The size of the message parameter is not accepted. The character string of the message must be terminated with a zero word (two ASCI NUL bytes positioned on a word boundary).

If a NUL byte appears in the message string in the right byte of a word, then the message will not have the task/overlay name and <HOLD message added.

Messages exceeding 52 characters are shortened so that the name of the task/overlay and HOLD messages appear on a 72-character line with the remaining part of the message. If the date/time stamp option is set, the system date and time are printed along with the message.

If the release resource option is set the task voluntarily gives its maps and register block back to the system while it is inactive. Consider using this option for high priority tasks that are inactive for long periods of time and/or are not critical at reactivation.

If this service is called from an OC overlay, the message is output to the CO file regardless of the setting of Bit 1 in Register 8.

USAGE EXAMPLES

Output a message to the operator and return control when the message is printed.

```
        LDI,R15    BUFFER                   BUFFER ADDRESS.
        LDI,R8     MESSAGE                  CALL NUMBER.
        REX,MAXIV                           REQUEST THE SERVICE.
         -
BUFFER  DFC        " GO TO LUNCH",0         MESSAGE WITH UPSPACE.
         -
```

Output message to operator, then suspend execution of calling task until operator releases or resumes task.

```
          LDI,R15       BUFFER              BUFFER ADDRESS.
          LDI,R8        MESSAGE.HOLD        CALL NUMBER + HOLD OPTION.
          REX,MAXIV                         REQUEST THE SERVICE.
           -
           -
BUFFER    DFC     "1MOUNT NEXT TAPE",0      MESSAGE WITH FORM FEED.
```

Output a message to the operator with a date/time stamp

```
          LDI,R15       BUFFER              BUFFER ADDRESS.
          LDI,R8        MESSAGE.STAMP       CALL + STAMP OPTION.
          REX,MAXIV
           -
           -
BUFFER    DFC           " SYSTEM TIME",0
```

Output to the operator:

```
12 JUN 84 12:30 15
SYSTEM TIME
```

SUSPEND CALLING TASK INDEFINITELY

Call Name:	WAIT	Call Number:	#11 (REX,MAXIV)
Option Names:	HOLD	Option Values:	#8000
	NOSEC		#4000
	RESOURCE		#2000
	STAMP		#1000

SERVICE PERFORMED

This service suspends execution of a task until resumed by a cooperating task, operator, timer, or interrupt. The HOLD option permits only the operator to resume the execution of the task, and types a short message on the operator's terminal that identifies the task entering the HOLD state:

/taskname/overlayname<HOLD

When a task is resumed by another task and the resumed task is already active, the resume is recorded and is considered a "secondary resume". therefore any task calling the WAIT service without the HOLD option does not WAIT if it has a secondary resume pending. This feature may be defeated by using the NOSEC option, which causes secondary resumes to be ignored.

If the RESOURCE option is set (= 1) the task voluntarily gives up its maps and register block while it is inactive. These resources are returned to the available pools for allocation to other tasks. (Refer to MESSAGE service.)

CALLING PARAMETERS

```
R8:    Service option and call number:
       Bit 0    HOLD option bit:
                =0 NO HOLD - but simple suspend desired.
                =1 HOLD
       Bit 1:   =0 Allows secondary resume
                =1 Defeats secondary resume requests.
                   Option name = NOSEC.
       Bit 2:   Resource specification
                =0 Task retains its resources.
                =1 Task voluntarily returns its resources
                   to the system while inactive.
                   Option name = RESOURCE.
       Bit 3.   Date/Time stamp.
                =0 Do not print date/time stamp with HOLD
                   message.
                =1 Print date/time stamp with the HOLD
                   message if the date/time stamp option
                   (DTS) was set at SYSGEN.
                   Option name = STAMP.
       Bits 4-7:  =0
       Bits 8-15: Service call number = #11 and call name = WAIT
```

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Service performed (no error conditions identified).

USAGE EXAMPLES

Suspend task execution indefinitely; anyone may resume this task.

```
        LDI,R8      WAIT          CALL NUMBER.
        REX,MAXIV                 REQUEST THE SERVICE.
          -
          -
```

Suspend my execution until operator decides to resume or release me.

```
        LDI,R8      WAIT!HOLD     CALL NUMBER + HOLD OPTION
        REX,MAXIV                 REQUEST THE SERVICE.
          -
          -
```

Suspend my execution indefinitely and ignore any pending secondary RESUME requests; anyone can resume me.

```
        LDI,R8      WAIT!NOSEC    CALL NUMBER + NOSEC OPTION.
        REX,MAXIV                 REQUEST THE SERVICE.
          -
          -
```

END EXECUTION OF CALLING TASK - NORMALLY

Call Name:	EXIT	Call Number:	#12 (REX,MAXIV)
Option Name:	KEPLOC	Option Value:	#8000
	NOSEC		#4000
	RETCODE		#2000

SERVICE PERFORMED

This service is called by a task when it has completed execution. All I/O operations that have been queued by the task continue until completed. At that time, all memory is deallocated, including the memory used for the Task Control Block (TCB) and task body (nonresident tasks only). The TCB is set not active and, if the task is a transient nonresident task and is not connected for subsequent execution by a timer or interrupt, the TCB is removed from the CPUQ. All local timers, which the calling task may have connected to perform a function on itself, are disconnected (including the Delay Timer). To defeat this automatic local timer disconnect, the EXIT service should be called with the KEPLOC option bit set. This service also releases any non-permanent exclusive use condition that this task might hold for a device. If the ACTIVATE service has been requested for a task, and the task is already active, the activate request is recorded and the task is immediately re-activated when calling the EXIT service. To defeat this secondary activation feature the NOSEC option should be used.

CALLING PARAMETERS

```
R8:     Service option and call number.
        Bit 0:      =0 Disconnect local timers
                    =1 Do not disconnect local timers.
                       Option name = KEPLOC.
        Bit 1:      =0 Allow secondary activations
                    =1 Defeat pending secondary activations.
                       Option name = NOSEC.
        Bit 2:      =0 No information returned to waiting task.
                    =1 Return information in R12-R15.
                       Option name = RETCODE.
        Bits 3-7:   =0
        Bits 8-15: Service call number = #12 and call name = EXIT
```

R12-R15: If Bit 2 of R8 is set, then the contents of these registers are returned to a waiting task. (This option is used with the Programmer's Workbench).

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	- Returned regardless.

PROGRAMMING CONSIDERATIONS

To exit a nonresident task without deallocating its resources, the ESTABLISH service can be used by the task or a cooperating task (for example, the operator). If an established nonresident task is deestablished when it is inactive, it immediately aborts and exits the system. Batch processing tasks reload their root task program (for example, JOB CONTROL) as an overlay when any overlay calls the EXIT service.

If the File Manager is configured and a FAT/LDT pair was used by the task but never closed, the EXIT service calls the File Manager CLOSE ALL service. Thus all File Manager files are properly closed during task EXIT.

When a Workbench task that has spawned other tasks exits, all of its spawned tasks are killed with an abort reason of PWB.

USAGE EXAMPLES

Terminate my execution and disconnect local timers.

```
         LDI,R8       EXIT             CALL NUMBER.
         REX,MAXIV                     REQUEST THE SERVICE
```

Terminate my execution but do not disconnect local timers.

```
         LDI,R8       EXIT!KEPLOC      CALL NUMBER + KEPLOC OPTION.
         REX,MAXIV                     REQUEST THE SERVICE.
          -
          -
```

Terminate my execution, disconnect local timers, and ignore any secondary activation requests pending.

```
         LDI,R8       EXIT!NOSEC       CALL NUMBER + NOSEC OPTION.
         REX,MAXIV                     REQUEST THE SERVICE.
          -
          -
```

TERMINATE EXECUTION OF CALLING TASK - ABNORMALLY

Call Name: ABORT Call Number: #13 (REX,MAXIV)

Option Name: WHERE Option Value: $0000

SERVICE PERFORMED

This service abnormally terminates the execution of the calling task upon entry to the service. All uncompleted I/O operations that were queued previously by the calling task are terminated without completion. The abort message to the "system" operator includes a 3-character "why" code, a three character "what" code, and a hexadecimal address (or other optional numeric information) indicating "where" the abortive action occurred in the executing task's addressing space. The system date and time are printed if the date time stamp option (DTS) was set at SYSGEN. The message is in the following format:

DD MMM YY HH:MM:SS
.ABORT(why what where*) /taskname/overlayname

To receive abort messages at a remote terminal, assign the local logical file CO to that remote device. Otherwise, messages appear on the system operator's terminal.

If the system option DUMP (Bit 15) is set to one in the task's TCB, then a printed hexadecimal dump is output to the DO (Diagnostic Output logical file (global or local)). This dump contains the PS (Program Status word) of the program and the contents of its general registers just prior to the abort call) followed by a dump of the entire addressing space(s) of the aborted program. Unallocated memory areas are indicated as such in the dump report. Refer to Appendix E entitled ERRORS AND ABORT REASON CODES in the MAX IV General Operating System, System Guide Manual.

After all the optioned abortive measures described above have been completed, the task is forced to call the EXIT service (#12), which completes the termination process appropriate to the task type (resident, nonresident, established, prescheduled). No return of control to the calling task occurs unless the task has been specially cataloged as being "unabortable" or "batch" using the PECULIAR statement of the TASK/OVERLAY CATALOGER. Refer to the TASK OVERLAY CATALOGER Programmer's Reference Manual listed in the Preface.

CALLING PARAMETERS

R0: Service call number and options.
Bit 0: =0 Virtual address of last REX call is to appear as the "where" address in the abort message

=1 "Where" address parameter passed in R15 is to appear in abort message. Option name = WHERE.

Bits 1-7: Reserved for system expansion

Bits 8-15: Service call number = #13 and call name = ABORT.

R13: The "why" reason code: This is a 3-character CAN-code) message to be conveyed to the "system" operator by the global (or local) CO file. All "why" reason codes should be unique in an operating system, thus the user should always pick reason codes for application programs that are not used by MAX IV. (Refer to Appendix E entitled ERRORS AND ABORT REASON CODES in the MAX IV General Operating System, System Guide Manual.)

R14: The "who/what" reason code: This is a 3-character CAN-code) message to be conveyed to the "system" operator by the global (or local CO file. This message may be used to add more information about the reason for the task abort.

(R15. The "where" address: This is an optional parameter that may be a virtual address (or other 16-bit numeric information, to be conveyed to the "system" operator by the global (or local) CO file This number is printed in hexadecimal notation and may be used for information about the reason for the task abort. This argument is ignored unless Bit 0 of the call options (R8 is set to 1. If this option is reset to 0, the virtual address printed points to the last REX service call executed by the aborting program (which will be tagged with an asterisk if that call was made from MAP 0 addressing space instead of from the instruction map (IM) of the calling task,. This normally means that an executive service, called by the task, has accessed arguments in the caller's addressing space that have violated access rights or that are unacceptable to the executive service.

CONDITION CODES UPON RETURN

None - service does not return control to calling task, unless it is a batch processing task or unabortable task and in such cases, the task is restarted.

PROGRAMMING CONSIDERATIONS

Any I/O operation queued by the calling task prior to calling this service is terminated without regard to completion or the state of the device.

Some system asks (for example, the OC task) can be aborted, without an ABORT message appearing on the system operator's terminal. Such message inhibition is controlled by a bit in the TCB of the task, which can only be manipulated by a privileged task or service called by a task.

To abort a task from an external point (for example, timer, interrupt, other tasks, use the KILL service (#17) or its scheduled equivalent in the CONNECT service (#18).

A REX service routine aborting the calling task can enter the service directly (BRU MSABOR, instead of using REX. In this case, the last 15 cells in the task's push stack are printed on the optional dump instead of the contents of the task's general registers and the last REX entry address is printed as the "where" address, unless the option (in R8) qualified as a "where" address.

If the File Manager is configured and a FAT/LDT pair was used by the task and never CLOSED, the ABORT process invokes the File Manager CLOSE ALL service.

When a Workbench task aborts that has spawned other tasks, all of its spawned tasks are aborted with a reason code of PWB.

USAGE EXAMPLES

I am a REX service executing in MAP Ø on behalf of a calling task in MAP 2. I have discovered that this task has passed me an illegal file name in CAN-code already in R14). I want to tell the system operator about this failure.

```
        ...
        LDI,R8   ABORT!WHERE  ABORT CALL WITH "WHERE" TO BE SUPPLIED.
        LDI,R13  @DUM         "WHY" CODE.
* ILLEGAL FILE NAME ALREADY IN R14 IN CAN-CODE (@fnm)
        LDM,R15  Ø            GET ADDRESS OF TASK'S ENTRY CALL TO ME
*                              (REX LINK).
        REX,MAXIV             REQUEST THE SERVICE.
*       ///////               NO RETURN.
```

The following message is printed on the CO file if the date/time stamp option (DTS) was set at SYSGEN.

```
      12 JUN 84 12:30:15
      !ABORT (DUM fnm XXXX*) /tname/oname
```

I am an application program that has encountered illegal data -- and I am not equipped to process it - this is a last resort!

```
*       ...
        LDI,R13       @GIV       CHOOSE "WHY" CODE.
        LDI,R14       @UP        CHOOSE "WHAT" CODE.
        LDI,R8        ABORT      SELECT ABORT SERVICE.
        REX,MAXIV                REQUEST THE SERVICE.
*       ///////                  NO RETURN.
```

The following message is printed on the CO file:

```
      12 JUN 84 12.30.15
      !ABORT (GIV UP  XXXX ) /tname/oname
```

DELAY

SET (GET) LOCAL DELAY TIMER (STATUS)

Call Name: DELAY Call Number: #14 (REX,MAXIV)

Option Names: QUICK Option Values: #8000
GETNOW #4000
RESOURCE #2000

Argument Option Name: ELAPSED Argument Option Value: #8000

SERVICE PERFORMED

This service uses the simple dedicated local timer that belongs to every normal task. A delay timer exists for all tasks. The service may set or reset the timer or may examine the current time remaining in the timer. The service may also be used to suspend the execution of the calling task until the time remaining in the timer has expired. Whenever the timer expires, a context switch "event" occurs and the task resumes. When this occurs, the task may decide to set the timer and suspend itself now, later, or never. If the task is forcefully resumed by another task, time scheduler, or interrupt scheduler, and it was suspended from within the DELAY service, then it will exit from the DELAY service with a non-zero time value returned in its registers. If the RESOURCE option is set, (=1), the task voluntarily gives up its maps and register block while inactive These resources are returned to the available pools for allocation to other tasks.

CALLING PARAMETERS

R0: Call options and call number:
Bit 0: Return mode for SET operation:
=0 Service suspends calling task after setting new time value into the timer.
=1 Service returns to calling task immediately after setting new time value into the timer. Option name = QUICK.
Bit 1: Set/Get option:
=0 Service sets new value into timer and responds according to Bit 0.
=1 Service gets current value in timer and return immediately to the calling task (Bit 0 is ignored . Option name = GETNOW.
Bit 2. Resource Specification:
=0 Task retains its resources.
=1 Task voluntarily returns its resources to the system while inactive. Option name = RESOURCE.
Bit 8-15: Service call number = #14 and call name = DELAY.

R14. Time mode and value (in minutes) to set timer, if Bit 1 of R8 is reset to 0

Bit 0: Time mode option bit:

=0 Time value is in Time-of-Day format. The Delay timer expires when this specific time-of-day arrives.

=1 Time value is in elapsed format. The Delay timer expires when this number of time counts have passed.

Bit 1-15: Number of minutes in time value to set timer in selected format.

R15. Time value in ticks (fractions of a minute) to set timer if Bit 1 of R8 is reset. This value of time is in addition to the minutes in R14. A "tick" is the smallest unit of time maintained by the real-time clock for the time-of-day clock, delay timers, and timers for task scheduling. A tick is normally some integral multiple of .005 seconds, as specified by a system generation parameter and is derived from a 200 Hz clock interrupt. A tick is .01 second (2 clock interrupt periods) on most standard systems. The maximum value of the argument is number-of-ticks-per-minute minus one.

CONDITION CODES UPON NORMAL RETURN

NZOC	Condition
'1000	Normal return 1 - service has performed its specified function.
'1001	Normal return 2 - service has performed its specified function, but task was not suspended as requested because it had been resumed while not suspended. Secondary resume was set.
'0000	Illegal time value arguments were passed.

REGISTERS CHANGED UPON RETURN

R14: the current elapsed minutes remaining in the Delay timer (positive).

R15: the current elapsed ticks remaining in the Delay timer (positive) in addition to the number of whole minutes in R14.

PROGRAMMING CONSIDERATIONS

Returned time counts are always in an elapsed format (counts remaining until timer expires) regardless of whether the original calling parameters used to set the timer were expressed in Time-of Day format or elapsed format. Thus, the service may be used as a "time-of-day" to "elapsed" conversion service by simply setting the timer with a "time-of-day" value and then reading out the returned elapsed result with the Quick Return option being specified. The reverse conversion can also be made by reading out the elapsed time

remaining in the timer, and then adding this elapsed time to the current Time-of-Day as obtained by calling the TIME service (#40 with a request for simple Time-of-Day return.

If the calling task is aborted or exits, then its delay timer is removed from the timer queue (if in use). If the task is a batch processing task or unabortable task, and restarts, its delay timer is expired.

USAGE EXAMPLES

Delay the execution of my program until at least 2.5 minutes have expired.

```
        LDI,R8      DELAY         CHOOSE DELAY SERVICE AND ITS OPTIONS.
        LDI,R14     2!ELAPSED     SET FOR 2 MINUTES (ELAPSED) AND .
        LDI,R15     500           SET 1/2 MINUTE IN TICKS ( 01 SEC/TICK).
        REX,MAXIV                 REQUEST THE SERVICE
*       .                         RETURN HERE 2.5 MINUTES LATER.
```

Set my delay timer for the next occurrence of 12 o'clock (noon) but do not suspend my program.

```
        LDI,R8     DELAY QUICK    CHOOSE DELAY SERVICE AND ITS OPTIONS.
        LDMD,R14   NOON           FETCH TIME-OF-DAY VALUE FROM MEMORY.
        REX,MAXIV                 REQUEST THE SERVICE.
*       ...                       RETURN HERE AS SOON AS TIMER IS SET.

NOON    DPC        12*60,0        NOON IN MINUTES AND TICKS IN OPERAND
                                  MAP.
```

Is it noon yet?

```
        LDI,R8     DELAY!GETNOW  SELECT DELAY SERVICE PLUS OPTION.
        REX,MAXIV                REQUEST THE SERVICE.
        TRRD,R14,R14             SET CONDITION CODES TO VALUE OF
*                                  TIMER COUNT.
        HZS,EXPIRED              HOP IF TIMER EXPIRED (IT'S NOON .
*        .                       NO, IT'S NOT NOON YET.
```

CONTINUE EXECUTION OF A SUSPENDED TASK

Call Name: RESUME — Call Number: #15 (REX,MAXIV

Option Name: RETTCB — Option Value. #2000

SERVICE PERFORMED

When a task calls a WAIT or DELAY service, it normally becomes suspended. If some other task calls this RESUME service on behalf of the suspended task, the suspended task is immediately resumed at the point where it was originally suspended.

CALLING PARAMETERS

R8: Call number and options.
- Bit 2: =0 Task Control Block (TCB) address is not returned
  - =1 Task Control Block (TCB address is returned. Option name = RETTCB.
- Bits 8-15: Service call number = #15 and call name = RESUME.

R14: First to third CAN-code characters of the name of the task to be resumed.

R15: Fourth to sixth CAN-code characters of the name of the task to be resumed.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0000	Task not in CPU queue.
'0001	Specified task is not active.

REGISTERS CHANGED UPON RETURN

Normal return: R15 = Address of the Task Control Block (TCB) address if optioned.

Abnormal return: R15 = Condition Codes.
Bits 0-11. Unused
Bits 12-15: Condition Codes

PROGRAMMING CONSIDERATIONS

Tasks that have been suspended and also put into the HOLD state cannot be resumed by this service. Only an operator directive can resume a task in the HOLD state.

If the desired task is not active, a "No-operation" is performed. If active but not suspended, the Secondary Resume Bit is set in the task that causes the next WAIT or DELAY service executed by that task to be ignored.

If the specified task cannot be found, a code is set in the task condition code to indicate the error.

The TCB address may only be obtained by a privileged task.

USAGE EXAMPLE

RESUME a task with the name TEST01:

```
        LDI,R14       @TES                 SET TASK NAME.
        LDI,R15       @T01
        LDI,R8        RESUME               CALL NUMBER.
        REX,MAXIV                          REQUEST THE SERVICE.
        HNR,ERROR                          IF ERROR HOP TO SUBROUTINE.
*       ...
```

START THE EXECUTION OF A TASK

Call Name: ACTIVATE   Call Number: #16 (REX,MAXIV)

Option Name: RETTCB   Option Value: $2000

SERVICE PERFORMED

This service causes a new task to be activated in MAX IV. Tasks activate at various priority levels. (Refer to Figure 2-1.)

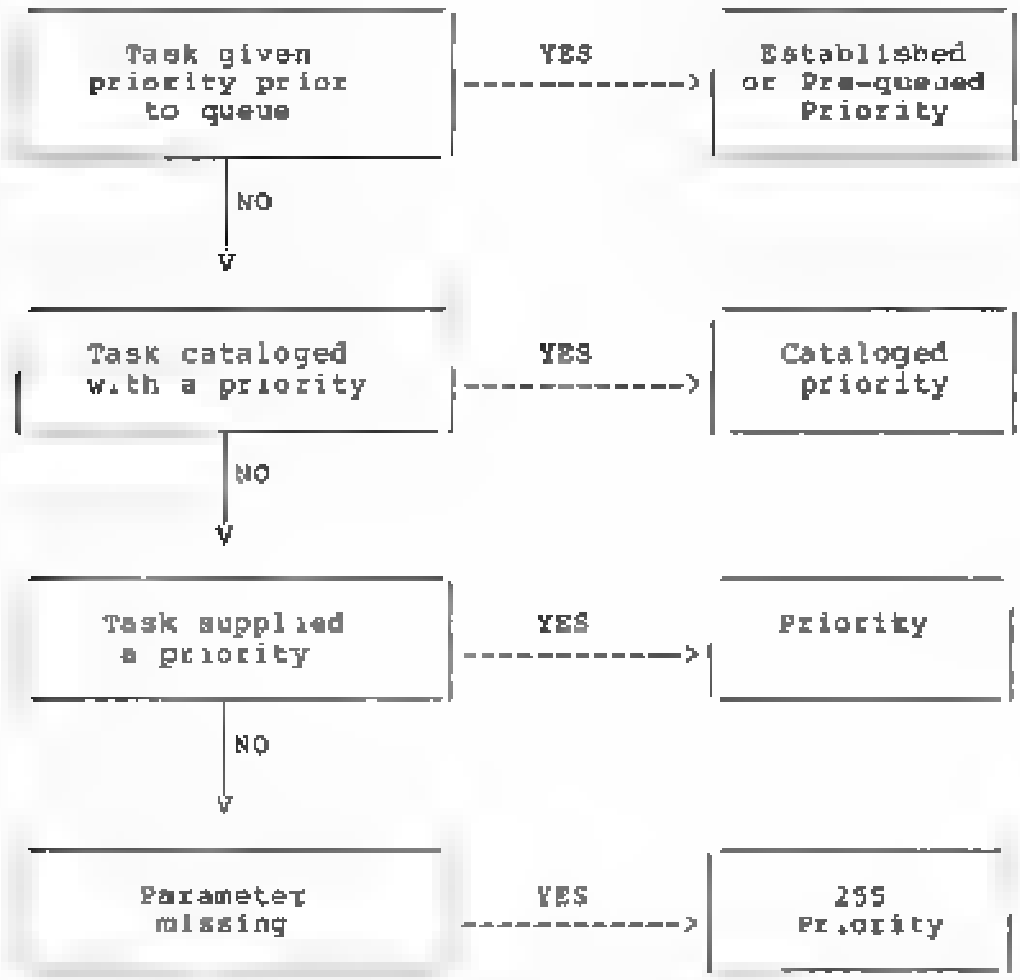

Figure 2-1. Priority Assignment During ACTIVATE Service

An error occurs if the invoking task has an influence limit that is "lower" (higher number) than the activated task's influence limit.

If the desired task is resident or established the file argument is ignored.

If the desired task is nonresident, a search of the directory for the global file specified is made. If the task is not found, the error is reflected in the condition codes. If the task is found, a Task Control Block (TCB) is queued and allocation of memory commences for loading of the task. If the task is a "prequeued" nonresident task, its TCB is already in the CPLQ and the priority argument is ignored.

CALLING PARAMETERS

R8: Service call number and options.
Bit 2: =0 Activated task's TCB not returned.
=1 Activated task's TCB returned. Option name = RETTCB.
Bits 8-15: Service call number = #16 and call name = ACTIVATE.

[R12]: Logical file name in CAN-code the file "LM" is chosen if R12=0 (the logical file name must be a GLOBAL file).

[R13:] Priority level 0=<P=<255 (if not established, or prequeued, or no priority specified when the task was cataloged).

R14: CAN-code characters 1-3 of the name of the task to be activated.

R15: CAN-code characters 4-6 of the name of the task to be activated

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0000	Task not queued and not found on disc file or, task not queued and MLSEQUENTIAL specified in SYSGEN.
'0001	Calling task has influence that is incompatible with specified task.
'0101	TCB too long, pages unavailable in the system.
'0110	Attempt to activate an overlay as a task. (Program was found on the file, but its format was incorrect.)
'0111	No TCB available.

REGISTERS CHANGED UPON NORMAL RETURN

R15: Address of TCB (if optioned). This address is virtual and refers to MAP Ø and is not necessarily the addressing space of the calling task.

REGISTERS CHANGED UPON ABNORMAL RETURN

R15. Condition Codes.

Bits Ø-11: Zero.
Bits 12-15. Same as NZOC.

PROGRAMMING CONSIDERATIONS

If the specified task cannot be found, or if a task is trying to activate a task with a higher influence limit, a code is set in the task condition code to indicate the error.

If the specified task is already active when activated, the Secondary Activation Bit is set in the task's TCB and the task will execute again after it exits the next time.

If the TCB address is requested to be returned it can only be used by a privileged task capable of addressing MAP Ø.

Tasks can only be activated from GLOBAL files.

Refer to the TASK/OVERLAY CATALOGER Programmer's Reference Manual listed in the Preface for more information on cataloging tasks.

USAGE EXAMPLE

Load Task with the name TESTØ1 from file XYZ and start its execution.

```
        LDI,R12     @XYZ        CHOOSE FILE NAME WHERE CATALOGED
        LDI,R13     255         SET PRIORITY LEVEL IF NOT CATALOGED.
        LDI,R14     @TES        FIRST PART OF TASK NAME.
        LDI,R15     @TØ1        SECOND PART OF TASK NAME.
        LDI,R8      ACTIVATE    SELECT SERVICE NUMBER.
        REX,MAXIV               REQUEST THE SERVICE.
        HNR,ERROR               IF ERROR, HOP TO A RECOVERY ROUTINE.
         ...
        ...
        ...
```

ABNORMALLY TERMINATE ANOTHER TASK

Call Name: KILL   Call Number: #17 (REX,MAXIV)

Option Name: RETTCB   Option Value: #2000

SERVICE PERFORMED

This service is similar to the ABORT service except that the calling task may cause another task to be aborted if the influence limit of the calling task permits it to operate on the other task. The following message is typed:

```
DD MMM YY HH:MM:SS
!ABORT (why what where*)/taskname/overlay name
```

This message includes a 3-character "why" code, a 3 character "what" code, and a hexadecimal address (or other optional numeric information indicating "where" the abort occurred in the aborting task's address space. The system date and time are always printed.

If the task being killed has a higher influence than the calling task the service is ignored and a condition code is set in the calling task indicating the error condition.

This service may be performed on a delayed basis by a connected interrupt or timer (see CONNECT service).

CALLING PARAMETERS

R0: Service Number and options:
- Bit 0: =0 Generate abort message
  - =1 Do not generate abort message.
- Bit 2: =0 TCB address not returned.
  - =1 TCB address returned. Option name = RETTCB.
- Bits 8-15: Service call number = #17 and call name = KILL.

R13: ABORT "why" reason code (in CAN-code).

R14: CAN-code characters 1-3 of the name of the task to be killed. If equal to zero, calling task is assumed.

R15: CAN-code characters 4-6 of the name of the task to be killed.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'1100	Operation successfully completed. (See Programming Considerations
'0000	Task not in CPUQ.
'0001	Tasks are incompatible.

REGISTERS CHANGED UPON RETURN

```
Normal return:    R15 = TCB address if optioned.
Abnormal return:  R15 = Condition Codes.
                        Bits 0-11:  Zero.
                        Bits 12-15: Same as NZOC.
```

PROGRAMMING CONSIDERATIONS

Only a privileged task capable of addressing MAP 0 can have the TCB address returned.

If condition codes = '1100 are returned, the task was in a temporary "cannot kill now" state or the TCB was not stable (due to the pending completion of an event such as task loading). However, TCBRIF is set to the KILL service, secondary activate is reset, and the KILL service will proceed as requested as soon as the temporary condition is cleared.

When a Workbench task that has spawned other tasks is killed, the spawned tasks are also killed with an abort reason of PWB.

USAGE EXAMPLE

Have my task terminate task "TEST01".

```
        LDI,R13        @KIL         ABORT CODE.
        LDI,R14        @TES         TASK NAME.
        LDI,R15        @T01
        LDI,R8         KILL         SERVICE NUMBER.
        REX,MAXIV                   REQUEST THE SERVICE.
        HNR,ERROR                   HOP IF ERR CONDITION.
         -
         -
         -
ERROR   EQU            $            ERROR ROUTINE
         -
```

2.4 TASK SCHEDULING SERVICES (#18-#1B)

CONNECT, TCONNECT

ALLOCATE SYSTEM TIMER TO SCHEDULE TASK CONTROL FUNCTION

Call Names:	CONNECT	Call Number:	#18 (REX,MAXIV)
	TCONNECT		#18
Option Name:	PERIODIC	Option Values:	#8000
	RECONNECT		#4000
	RETTCB		#2000
	XKILNAB		#700
	XKIL		#300
	XACT		#200
	XRES		#100

SERVICE PERFORMED

This service may be called by any task to schedule a RESUME service, an ACTIVATE service, or a KILL service to be performed on a delayed basis for itself or another task. This service basically allocates a system timer and prepares the desired task's TCB (putting it in the CPUQ if necessary) if the ACTIVATE service is requested. The calling task need not remain active (unless desired) after starting the task scheduling service for the specified task. Task scheduling services are performed when the assigned system timer has expired (one-shot or periodic). If the specified task cannot be found in the system or in the directory of the specified file or the number of timers is exhausted, this service is ignored and an error code is returned to the calling task. If the target task has a higher influence than the calling task, the service is ignored and an error code is set.

Upon calling the CONNECT service, the list of available system timers is scanned. If a vacant timer is found, it is set NOT AVAILABLE and a password that identifies it uniquely is returned to the calling task. This password must be used by the calling task to enable (THAW), disable (FREEZE) or deallocate (UNCONNECT) the timer. This service exits with the allocated timer disabled and THAW must be used to begin the timing operation. (See EXIT Service for disconnect on Task Exit.)

CALLING PARAMETERS FOR TIMERS

[R6:] Period minutes (if PERIODIC optioned):
Bit 0: =0 Value is time-of-day format.
=1 Value is elapsed time.

[R7:] Period ticks (if PERIODIC optioned).

R8: Service options and call number:
Bit 0: =0 Timer is one-shot.
=1 Timer is periodic (R6 & R7 contain period).
Option name = PERIODIC.

Bit 1: =0 Allocate new timer.
=1 Reconnect Timer (Password in R9). Option name = RECONNECT.

Bit 2: =0 Do not return my TCB address in R15.
=1 Return my TCB address in R15. Option name = RETTCB.

Bit 3: =0 Must be reset for Timer CONNECT. (See ICONNECT - #1018.)

Bits 4-7: = Function code
- =0 Custom event.
- =1 RESUME. Option name = XRES.
- =2 ACTIVATE. Option name = XACT.
- =3 KILL. Option name = XKIL.
- =7 KILL but do not print abort message. Option name = XKILNAB

Bits 8-15: Service call number = #18 and call name = CONNECT or TCONNECT.

[R9:] Timer Password (if timer was allocated).

R10. Initial minutes to set timer for first count-down:
Bit 0: Format of Initial minutes
=0 Time-of day format.
=1 Elapsed time.

R11: Initial ticks to set timer if fractions of a minute are to be specified.

R12: Logical file name of target task (where task cataloged (ACTIVATE ONLY).

R13: Priority level of target task (ACTIVATE ONLY) if not installed with cataloged task.

R14: Task name (first 3 CAN-code characters) to be the target of a scheduled service

R15: Task name (last 3 CAN-code characters) to be the target of a scheduled service

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0000	Task not found.
'0001	Tasks are incompatible.
'0011	Timer unavailable.
'0100	Illegal parameter.
'0111	No TCB available.

REGISTERS CHANGED UPON NORMAL RETURN

R9: Timer password (if one was allocated).

R15. TCB address (if optioned).

PROGRAMMING CONSIDERATIONS

Only a privileged task can have its TCB address returned.

USAGE EXAMPLE

CONNECT task TEST01 to an available timer. ACTIVATE task after 500 ticks (5 seconds) initial delay and re-activate every 5 minutes thereafter.

```
ELAPSE  EQU        #8000              ELAPSED TIME MODE.
*        . .
        LDI,R6     5+ELAPSE           REPEAT EVERY 5 MINUTES ...
        ZRR,R7                        ... AND 0 TICKS.
        LBR,R10,B0                    SET TO 0 MINUTES (ELAPSED ...
        LDI,R11    500                .. AND  500 TICKS INITIALLY.
        LDI,R12    @LM                FILE NAME.
        LDI,R13    200                PRIORITY, IF NEEDED
        LDI,R14    @TES               TASK NAME.
        LDI,R15    @T01               TASK NAME.
        LDI,R8     CONNECT+PERIODIC   SERVICE NUMBER AND OPTIONS.
        REX,MAXIV                     REQUEST THE SERVICE.
        HNR,ERROR                     HOP IF ERROR
* SAVE R9 FOR USE IN THAW SERVICE TO ENABLE TIMER
          .
          .

ERROR   EQU        $                  ERROR ROUTINE STARTS HERE.
          .
          .
          .
```

ALLOCATE INTERRUPT TO SCHEDULE TASK CONTROL FUNCTION

Call Name.	ICONNECT	Call Number.	#18+#1000 ,REX,MAXIV,
Option Names:	PERIODIC	Option Values:	#8000
	RETTCB		#2000
	XKILNAB		#700
	XKIL		#300
	XACT		#200
	XRES		#100

SERVICE PERFORMED

The specified hardware level is allocated and connected to perform the delayed service for the specified task. The possible service may be ACTIVATE, KILL, or RESUME. These delayed services are performed when the assigned hardware interrupt is triggered by an internal or an external event.

If the specified task cannot be found in the system or in the directory of the specified file, or if the specified interrupt level is already in use, this service is ignored and an error code is set in the calling task condition code. If the target task has a higher influence than the calling task, the service is ignored and the error condition is reflected in the calling task condition code.

Hardware interrupt levels are directly addressed (Ø<(R9 <255) rather than allocated from a pool since a known hardware mechanism must be installed uniquely and connected to such levels. Levels #7-#B are true priority levels and levels 32-63 are special party-line levels implemented through a digital I/O subsystem (I/OIS). Levels 64-255 are special levels implemented through I/OIS common alarm input channel option. Hardware interrupt and trap levels dedicated for use by MAX IV may not be addressed by this service.

CALLING PARAMETERS

R8: Service options and call number:

Bit 0:	=0	Interrupt is disabled when scheduled function performed (one-shot).
	=1	Interrupt remains enabled after scheduled function performed. Option name = PERIODIC.
Bit 2:	=0	Do not return my TCB address in R15.
	=1	Return my TCB address in R15. Option name = RETTCB.
Bit 3:	=1	Schedule interrupt instead of timer (see previous service).

Bits 4-7: = Function code:
=0 Custom event to be executed by interrupt activation. Refer to U.IEVT in the USER HOOKS chapter of the MAX IV Data Structures, SDM.
=1 RESUME service to be executed by interrupt. Option name = XRES.
=2 ACTIVATE service to be executed by interrupt. Option name = XACT.
=3 KILL service to be executed by interrupt activation. Option name = XKIL.
=7 KILL service. Abort message is not to be printed. Option name = XKILNAB.

Bits 8-15: Service call number = Ø18 and call name = ICONNECT.

R9: Interrupt level.

R12: Logical file name (ACTIVATE only) from which task is to be loaded.

R13: Priority Level (ACTIVATE only) if not cataloged with task.

R14: Task name (first 3 CAN-code characters) to be scheduled by interrupt. If equal to zero, the service is performed on the calling task.

R15. Task name (last 3 CAN-code characters to be scheduled by interrupt.

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	Operation completed successfully.
'ØØØØ	Task not found.
'ØØØ1	Tasks are incompatible.
'ØØ11	Interrupt level unavailable.
'Ø1ØØ	Illegal parameters.
'Ø111	No TCB available.

REGISTERS CHANGED UPON NORMAL RETURN

R15: TCB address (if optioned).

USAGE EXAMPLE

Allocate interrupt level 7 to RESUME task TESTØ1 when interrupt level 7 is enabled (thawed and its request latch is subsequently triggered.

```
        LDI,R9       7                    INTERRUPT LEVEL.
        LDI,R14      @TES                 TASK NAME.
        LDI,R15      @TØ1
        LDI,R8       ICONNECT!XRES        CALL NUMBER AND OPTION.
        REX,MAXIV                         REQUEST THE SERVICE.
        HNR,ERROR                         HOP IF ERROR.
         -
         -
         -
ERROR   EQU          $                    ERROR ROUTINE STARTS HERE.
         -
         -
```

DISABLE AND DEALLOCATE TASK SCHEDULER

```
Call Names:   UNCONNECT          Call Numbers:  $19    (REX,MAXIV,
              TUNCONNECT                        $19
              IUNCONNECT                        $1019
```

SERVICE PERFORMED

This service disables an interrupt or timer. It is then marked vacant or available and free to be allocated to another user.

CALLING PARAMETERS

```
R0:  Service number and options:
     Bit 3:      =0 Disable timer.  Service call names are
                    UNCONNECT and TUNCONNECT.
                 =1 Disable interrupt.  Service call name =
                    IUNCONNECT.
     Bits 8-15: Service call number = $19.

R9:  Timer password or interrupt level
```

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0100	Illegal password or interrupt not found

USAGE EXAMPLES

Freeze and Deallocate timer TMA

```
       LDI,R9      @TMA          LOAD TIMER PASSWORD.
       LDI,R8      UNCONNECT     SERVICE NUMBER FOR TIMERS.
       REX,MAXIV                 REQUEST THE SERVICE.
       HNR,ERROR
         .
ERROR  EQU         $             ERROR ROUTINE.
         -
```

Freeze and deallocate level 7 interrupt.

```
       LDI,R9      7             LOAD INTERRUPT LEVEL.
       LDI,R8      IUNCONNECT    SERVICE NUMBER FOR INTERRUPTS.
       REX,MAXIV                 REQUEST THE SERVICE.
       HNR,ERROR                 HOP IF ERROR.

       ERROR EQU   $             ERROR ROUTINE.
         -
```

ENABLE ALLOCATED TASK SCHEDULER - TIMER OR INTERRUPT

Call Names:	THAW	Call Numbers:	$1A (REX,MAXIV
	TTHAW		$1A
	ITHAW		$101A
Option Names:	PERIODIC	Option Values:	$8000
	NEWPAR		$4000
	REMCON		$800

SERVICE PERFORMED

This service enables any previously allocated timer or interrupt scheduler that was set up by the CONNECT service (TCONNECT or ICONNECT). A timer begins counting with the next system clock interrupt while an interrupt level has its request latch cleared (past history reset) and then its enable latch set. Expiration of the timer count causes the scheduled task service to be performed while the subsequent setting of an interrupt level's request latch causes its scheduled task service to be performed. If either the timer or interrupt level was originally connected as a "periodic" scheduler, it will remain connected to the task after the first use of the scheduled service. A "periodic" scheduler must be explicitly disabled (FROZEN) by a task or it continuously schedules its service for the connected task. If the timer or interrupt was not "periodic" it is said to be a "one-shot" scheduler. Such a scheduler is usually "unconnected" from the scheduled task after the initial scheduled function is performed.

This service may be used to simply enable a scheduler with parameters that were pre-specified at the time it was connected; or the service may be made to bind new parameters and characteristics to the scheduler. In particular, a new "quality" may be given to a "one-shot" timer that causes it to become disabled when the scheduled service is first performed, but to remain connected to the task. Such a scheduler need only be re-enabled for it to become an active scheduler again. A "one-shot" scheduler may be made "periodic", and vice versa. For timer schedulers, new time settings may be specified.

CALLING PARAMETERS

```
[R6:]   Period minutes (if PERIODIC optioned):
        Bit 0:  =0 Value is time-of-day format.
                =1 Value is elapsed time.
[R7:]   Period ticks (if PERIODIC optioned).
```

R8: Call number and option bits:
Bit 0: New "periodicity" parameter if Bit 1 of R8 is set to one and:
=0 Scheduler is to become non-periodic that is, schedule is to be disabled but remain connected after initial execution of the scheduled function).
=1 Scheduler is to become periodic (that is, scheduler is to remain enabled after initial execution of the scheduled function). Option name = PERIODIC.
Bit 1. Determines if old or new parameters are to be bound and:
=0 Scheduler should be thawed using the parameters that were specified when connected.
=1 Scheduler should be thawed and new parameters are to be specified for scheduler that override those specified when connected (see Bits 0 and 4 of R8 and R6, R7, R10, and R11). Option name = NEWPAR.
Bit 3: Selects whether timer or interrupt scheduler to be enabled and:
=0 Timer scheduler is to be thawed (use R6, R7, R10, R11 and Bits 0 and 4 of R8 if new parameters supplied).
=1 Interrupt scheduler is to be thawed (use Bits 0 and 4 if new parameters supplied). Service call name = ITHAW.
Bit 4. New "one-shot" permanence parameter if Bit 1 of R8 is set to one, and:
=0 Scheduler is not "periodic", and scheduler is unconnected upon single execution.
=1 Scheduler is not "periodic", and scheduler remains connected upon single execution. Option name = REMCON.
Bits 8-15: Service call number = #1A and call name = THAW and TTHAW.

R9: Scheduler select argument.
0 = Timer password if timer scheduler to be thawed.
1 = Interrupt level number if interrupt scheduler to be thawed.

R10: Initial minutes to set timer for first count-down:
Bit 0 = Format of initial minutes
=0 Time-of-day format.
=1 Elapsed time.

R11: Initial ticks to set timer if fractions of a minute are to be specified

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0100	Specified scheduler is not connected, timer password not correct, or level number not legal.
'0000	Operation unsuccessful.

USAGE EXAMPLES

A previously connected timer's password has been stored in a memory cell, and I am ready to start timing. All the parameters that were specified at "connect" time should be used.

```
        LDM,R9     PASSWD       FETCH PASSWORD THAT CONNECT SERVICE
*                                 GAVE ME.
        LDI,R8     TTHAW        THAW TIMER AND USE OLD PARAMETERS
        REX,MAXIV               REQUEST THE SERVICE.
        HNR,ERROR               IF ERROR, HOP TO SERVICE ROUTINE.
*       ...                     RETURN HERE IF TIMER THAWED.

PASSWD DEC   0                  STORAGE CELL FOR PASSWORD.
ERROR  EQU   $                  ERROR ROUTINE REQUESTS HERE.
```

I have previously connected a "one shot" interrupt level (#7, and am ready to enable it, but I will make the scheduler remain connected after the function executes.

```
        LDM,R9     #7           SELECT INTERRUPT LEVEL TO THAW.
        LDI,R8     ITHAW!NEWPAR!REMCON
*                               SELECT THAW-INTERRUPT SERVICE AND
*                                 OPTIONS.
       REX,MAXIV                REQUEST THE SERVICE.
       HNR,ERROR                HOP IF ERROR.
*       ...
```

I have previously used a "one-shot" timer but I now want to set it to periodic and to set new initial and period time values as I THAW it.

```
        LDM,R9     PASSWD       FETCH PASSWORD OF ALLOCATED TIMER.
        LDI,R8     TTHAW NEWPAR!PERIODIC
*                               SET SERVICE AND OPTIONS
        LDMD,R6    NEWPER       NEW PERIOD TIME VALUE.
        LDMD,R10   NEWINI       NEW INITIAL TIME VALUE.
        REX,MAXIV               REQUEST THE SERVICE.
        HNR,ERROR               HOP IF ERROR
*
```

FREEZE, TFREEZE, IFREEZE

DISABLE CONNECTED TIMER OR INTERRUPT

Call Names:	FREEZE	Call Numbers:	#1B (REX,MAXIV)
	TFREEZE		#1B
	IFREEZE		#101B

SERVICE PERFORMED

This service disables any timer or interrupt connected to a task. The scheduler is not made vacant for re-assignment by this service but may again be "THAWED" or "RETHAWED" if desired. A frozen timer or interrupt is not available for re-assignment to other tasks until it is "unconnected". An exception is a one-shot timer or interrupt. Such a scheduler freezes and unconnects itself after expiration of its initial delay unless it is explicitly made to remain connected by the THAW service.

CALLING PARAMETERS

R8: Call number and option bits:
- Bits 3: =0 Disable timer. Service call name = FREEZE AND TFREEZE
  - =1 Disable interrupt. Service call name = IFREEZE.
- Bits 8-15: Service call number = #1B.

R9: Timer password or interrupt level of scheduler to be frozen.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0100	Timer or interrupt not connected.

REGISTERS CHANGED UPON NORMAL RETURN

None

USAGE EXAMPLES

Freeze previously connected hardware interrupt level stored at location INTER.

```
        LDM,R9          INTER        LOAD INTERRUPT LEVEL
        LDI,R8          IFREEZE      CALL NUMBER FOR INTERRUPTS.
        REX,MAXIV                    REQUEST THE SERVICE.
        HNR,ERROR                    HOP IF ERROR.
         .
         .
         .
ERROR   EQU             $            ERROR ROUTINES
         .
         .
         .
INTER   DFC             0
         .

         .
```

Freeze system timer stored at TIMER that I or somebody connected earlier.

```
        LDM,R9          TIMER        LOAD TIMER PASSWORD.
        LDI,R8          TFREEZE      CALL NUMBER FOR TIMER.
        REX,MAXIV                    REQUEST THE SERVICE.
        HNR,ERROR                    HOP IF ANY ERROR.
         .
         .
         .
ERROR   EQU             $            ERROR ROUTINE STARTS HERE.
         .
         .
         .
TIMER   DFC             0            TIME PASSWORD STORED HERE.
```

2.5 PRIORITY CHANGE SERVICE (#1C)

CHANGE

CHANGE PRIORITY OF SPECIFIED TASK

Call Name: CHANGE   Call Number: #1C (REX,MAXIV)

Option Name: RETTCB   Option Value: #2000

SERVICE PERFORMED

This service changes the current priority of an already scheduled or active task. A task can change the priority of another task only if it has a compatible influence limit.

CALLING PARAMETERS

R8: Call number and options:
  Bit 2: =0 TCB of specified task is not to be returned
       =1 TCB of specified task is to be returned. Option name = RETTCB.
  Bits 8-15: Service call number = #1C and call name = CHANGE.

R13: Bits 0-7: Must be zero.
  Bits 8-15: New priority level (0<P<255) to be specified. No level "higher" than the influence limit of the calling task will be permitted. Level 0 is the highest level of priority and influence.

R14: CAN-code characters 1-3 of the name of the task to be changed. If equal to zero, the calling task is assumed.

R15: CAN-code characters 4-6 of the name of the task to be changed.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0000	Task not found in CPU queue.
'0001	Tasks have incompatible influence levels.
'0010	New priority is higher than influence limit of specified task or R13 is not in the range of 0 to 255 inclusive.

REGISTERS CHANGED UPON NORMAL RETURN

R15: TCB Address if optioned.

REGISTERS CHANGED UPON ABNORMAL RETURN

```
R15:  Condition Codes
      Bits  0-11:  Zero.
      Bits 12-15:  Same as NZOC.
```

PROGRAMMING CONSIDERATIONS

This service affects the priority of any specified resident, prequeued, connected, or currently active task. It will not affect the cataloged priority of a transient nonresident task.

The effects of this service last only while the affected task remains in the CPU queue.

A returned TCB address can only be used by a privileged task capable of addressing MAP 0.

USAGE EXAMPLE

Change the priority level of the task "TEST01" to level 200. This assumes my influence limit is 200 or higher.

```
        LDI,R13        200              NEW LEVEL.
        LDI,R14        @TES             TASK NAME.
        LDI,R15        @T01
        LDI,R8         CHANGE           CALL NUMBER.
        REX,MAXIV                       REQUEST THE SERVICE.
        HNR,ERROR                       HOP IF ERROR.
         -
         -
```

2.6 TASK RELINQUISH SERVICES ($1D-$1E)

RELINQUISH

--- -- ------------------------ ----------------------------

LET LOWER PRIORITY TASKS USE CPU TILL NEXT SYSTEM EVENT
LET HIGHER PRIORITY TASKS KNOW CALLING TASK HAS CAUSED SYSTEM EVENT

------------------- --

Call Name:	RELINQUISH	Call Number:	$1D (REX,MAXIV)
Option Names:	EVENTSCAN	Option Values:	$8000
	RETLOCK		$8000
	RESOURCE		$2000

SERVICE PERFORMED

This service returns control to the caller at the completion of the next system event. Thus it provides a way of relinquishing control to a lower priority task while the calling task remains active. Another form is used to inform another (possibly higher priority) RELINQUISHING task that may be waiting for a particular event.

The RELINQUISH service, with the option bit reset to zero should be used by all tasks that become wait-bound in order that other tasks may use CPU time gainfully. A more "educated" form of the service is RELTILL ($1E).

If the resource option is set, (='1), the task will voluntarily give up its maps and register block while it is inactive. These resources will be returned to the available pools for reallocation to other tasks.

CALLING PARAMETERS

R0: Service option and call number:
- Bit 0: =0 Time is relinquished to lower priority tasks, although the calling task remains active. The TASKMASTER starts its scan at next lower priority task in CPUQ.
  - =1 All priority levels are reviewed by an event scan. The TASKMASTER starts its scan with the highest priority task. Option names = EVENTSCAN and RETLOCK.
- Bit 2: Resource specification where ...
  - =0 Task retains its resources.
  - =1 Task voluntarily returns its resources to the system while it is inactive. Option name = RESOURCE
- Bits 8-15: Service call number = $1D and call name = RELINQUISH.

CONDITION CODES UPON RETURN

NZOC	Condition
*1000	Normal return

USAGE EXAMPLES

Let lower priority tasks use CPU time. Return after next system event when the TASKMASTER finds me to still be the highest priority task.

```
        LDI,R0      RELINQUISH  SERVICE NUMBER.
        REX,MAXIV               REQUEST THE SERVICE.
*       ...                     RETURN HERE AFTER NEXT EVENT IF STILL
*                                 HIGHEST PRIORITY.
```

I have caused an event for which some other task might be waiting. Trigger the TASKMASTER to perform an event scan. Return after causing context switch as soon as I am highest priority executable task.

```
        LDI,R0     RELINQUISH!EVENTSCAN
*                               CALL NUMBER + OPTION.
        REX,MAXIV               REQUEST THE SERVICE.
*       ...                     RETURN HERE WHEN I'M HIGHEST AGAIN.
```

RELTILL

LET LOWER PRIORITY TASK USE CPU UNTIL SOME CONDITION IS SATISFIED

Call Name:	RELTILL	Call Number:	$1E (REX,MAXIV)
Option Name:	RETLOCK RESOURCE	Option Value:	$8000 $2000

SERVICE PERFORMED

This service is called when a task has no more useful work to do until some event (for example, another task, interrupt, timer expiration, or operator intervention occurs. A memory cell is used as a communication cell. When this cell is modified by the system event in a manner specified by the calling task, the calling task is allowed to exit this service. The condition may be to relinquish until either:

- A bit is reset or set in memory
- A memory value equals or does not equal another memory value
- A memory value equals or does not equal a register value.

During the indefinite waiting period, the calling task remains active and retests the specified criteria on every system event to see if it is the event it is waiting for. If not, the service again makes the calling task relinquish time to lower priority tasks until the next event occurs.

Often the calling task must perform a function, which cannot afford to be interrupted, after the condition is satisfied. If this is the case, the option bit of the service may be set when it is called which causes the service to exit with interrupt level 4 active. Thus, the calling task can follow the service call with coding that is NON-INTERRUPTABLE, but the caller must be sure to release the lockout level before proceeding very far. This "EXIT-WITH-LOCKOUT" option is ignored for unprivileged tasks since such tasks cannot release the lockout condition that requires the execution of a privileged machine instruction (RIA,4).

If the resource option is set, (='1), the task will voluntarily give up its maps and register block while it is inactive. These resources will be returned to the available pools for allocation to other tasks.

CALLING PARAMETERS

```
R0:   Service option and call number:
      Bit 0:    =0  Return unlocked
                =1  Return with Level 4 active (if privileged).
                    Option name = RETLOCK.
      Bit 2:    Resource specification.
                =0  Task retains its resources.
                =1  Task voluntarily returns its resources
                    to the system while it is inactive.
                    Option name = RESOURCE.
      Bits 8-15:    Service call number = $12 and call name =
                    RELTIL.

R2:  Address of memory cell to be tested.

R15: Selects the criteria for testing the specified cell and...
     Bit 0.   =0  Bit test
              =1  Value test
     Bit 1:   =0  Relinquish until bit reset or value equal
                  memory cell
              =1  Relinquish until bit set or value not equal
                  memory cell
     Bit 2:   =0  Value comparand address in another register
                  R(n
              =1  Value comparand in another register R(n)
  Bits 12-15: =   Bit number or comparand register number
                    (0<=N<=15 .
```

CONDITION CODES UPON RETURN

NZOC	Condition
*1000	Normal return.
*0100	Return with Level 4 active specified but user not privileged
*0011	Address in nonexistent memory
*0001	Access rights violation

PROGRAMMING CONSIDERATIONS

If MAP 0 is in the operand map in use at the time the REX service is called, the memory cell s) is assumed to be in MAP 0 ,otherwise assumed to be in the Task's operand map).

USAGE EXAMPLE

Relinquish until Bit 15 of the first word of my UFT resets.

```
        LDI,R2          UFTADR       LOAD UFT ADDRESS.
        LDI,R8          READIQUICK
        REX,MAXIV
         .
         :
        LDI,R2          UFTADR       LOAD UFT ADDRESS.
        LDI,R15         15           RELINQUISH UNTIL BIT IS RESET.
        LDI,R8          RELTILL      CALL NUMBER.
        REX,MAXIV                    REQUEST THE SERVICE.
*       ...                          RETURN HERE WHEN BIT RESET.

UFTADR  DEC             0,@SI,....   UFT LOCATIONS.
*       ..
```

2.7 JOB/TASK STATE SERVICE ($20)

SET OR GET JOB STATES OF CALLING TASK

Call Name: STATE   Call Number: $20 (REX,MAXIV

Option Name: MODSTATE   Option Value: $0000

SERVICE PERFORMED

The STATE service can be used to set selected job or task states or to retrieve these states, including Pipeline Mode.

CALLING PARAMETERS

```
R0:   Call number and option bits:
      Bit 0.       =0 Retrieve task states.
                   =1 Set task states.
                      Option name = MODSTATE.
      Bits 1-6:    =0
      Bit 7:       =0 State service
                   =1 Pipeline mode service
      Bits 8-15: Call Number = $20, and Call Name = STATE
```

<u>If R0, Bit 7 is reset (=0)</u>

Select Job States (Word EXTJST in TCB Extension EXT) of a Task

(R15). The following states are retrieved from or put into the Job Status Word (EXTJST) depending on the Bit 0 setting.

- Retrieved states (if R0 Bit 0=0)   or
- Set states (if R0 Bit 0=1)

```
Bit 1:  Floating Point Overflow History.
Bit 2:  Floating Point Underflow History.
Bit 3:  "GO" error has occurred
Bit 4:  Decimal Arithmetic Exception History.
Bit 5:  Decimal Data Exception History.
```

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return.

REGISTERS CHANGED UPON NORMAL RETURN

R15: Old states that were loaded from the state word. If Bit 0 of R0 is set, then it is the state word just prior to setting the new states.

If R8, Bit 7 is set (=1)

Retrieve current pipeline mode state (if R8 Bit 0=0 or
Select new pipeline execution mode state (if R8 Bit 0=1)

R15, Bit 15 =0 Task is to exit pipeline mode.
=1 Task is to enter pipeline mode.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Task was in non-pipeline mode.
'1001	Task was in pipeline mode.

REGISTERS CHANGED UPON NORMAL RETURN

R15, Bit 15 =0 if the task was in non-pipeline mode.
=1 if the task was in pipeline mode.

USAGE EXAMPLE

Get the Job States of the calling task.

```
        LDI,R8        STATE      CALL NUMBER.
        REX,MAXIV                REQUEST THE SERVICE
*       ...                      RETURN HERE WITH JOB STATES IN R15.
```

SPAWN A TASK

Call Name.	SPAWN	Call Number:	#0022 ,REX,MAXIV
Option Names:	LDFILE AOE	Option Values:	#8000 #4000

SERVICE PERFORMED

SPAWN provides the capability of creating a task from another task running in the system. This service may be used by any task that was cataloged with TOC using the WORKBENCH option or by a task that was SPAWNed. This service provides the following capabilities:

- o The resources of the SPAWNed task are the same as those of the SPAWNer.
- o The body of the SPAWNed task can be a duplicate of that of the SPAWNer, or it can be an overlay loaded from disc
- o The SPAWNer can specify the priority at which the SPAWNee is to execute.
- o The SPAWNer can specify whether in the case of errors during SPAWNing principally resources that could not be duplicated) the SPAWNee is to be aborted or not.

CALLING PARAMETERS

R8: Call number and option bits:

Bits 0 3:	= '1000	SPAWN a task with my resources and load the specified overlay for that task. Option name = LDFILE.
	= '0100	Abort SPAWNed task if error in resource duplication occurs. Option name = AOE.
Bits 4-7:	= '0000	To indicate SPAWN option within Programmer's Workbench REX's
Bits 8-15:	Service call number = #22 and call name = SPAWN.	

R12: LFM of file containing body of task to be SPAWNed (specified only for LDFILE option)

R13: Execution priority of SPAWNed task -1 means default priority to the priority of the SPAWNer

R14-15. Name of overlay to be loaded with SPAWNed task specified only for LDFILE option)

REGISTERS RETURNED

Normal Return

R14-15. The SPAWNed task's name

Error Return

R15. The condition codes

CONDITION CODES UPON RETURN

NZOC	Condition
1000	Successful SPAWN
1100	Task SPAWNed but not all resources were duplicated
1010	You are the SPAWNee that was just SPAWNed
1110	You are the SPAWNee but not all resources were duplicated
0000	Module not found on disc or MLSEQUENTIAL specified in SYSGEN.
0001	SPAWNer task's influence limit is incompatible with specified SPAWNee's priority
0010	SPAWNer name is in illegal format
0011	WORKBENCH not SYSGENed or Task not TOC'ed WORKBENCH.
0100	Too many SPAWNee's
0101	TCB too long, pages unavailable in system
0111	Not enough system resources available to SPAWN a task (TCB, Transient LDT, Actual Memory, Virtual Memory

PROGRAMMING CONSIDERATIONS

- o The resources and options in the prologue will be duplicated. Status flags will be set appropriately and pointers to other fields will be built as needed.
- o The resources specified in the EXT will be duplicated. Pointers and save areas will be built as necessary
- o A new service buffer extension will be built if the SPAWNer had one.
- o The push stacks will be built empty or duplicated, depending on whether the SPAWNee's body is a duplicate of the SPAWNer or is loaded from disk.
- o The FAT will be duplicated. Any device that is under exclusive use by the SPAWNer will not be available to the SPAWNee until a GIVE is performed.

- For File Manager files, duplicate FCBs will be built. Any file that is OPEN with exclusive use by the SPAWNer will not be passed to the SPAWNee. This case will be considered an error for the abort on error option. If the option is not selected, both the SPAWNer and the SPAWNee will receive condition codes back indicating that not all resources were passed (it may not be possible to pass this to the SPAWNee if the SPAWNee's body was not duplicated).

- The variables extension will be duplicated

- The loader extension will be built as appropriate. In the case of a "duplicate" SPAWN, the SPAWNee's loader extension will be a duplicate of the SPAWNer. In the case of a "load from disk" SPAWN, the SPAWNee's loader extension will be built with values appropriate for the disk load, and with no trap table.

- The delay extension will be duplicated.

- The page-sharing extension will be duplicated if the SPAWN is a duplicate SPAWN. Otherwise, it will be built as appropriate for the disk load.

- The system page list will be duplicated.

- The III allocation extension will be duplicated.

- The SPAWNer's task name must be a TMP style name; that is, the last 3 characters of the name must be numeric.

- The SPAWNed task's name will be generated from the SPAWNer's task name. The first 3 characters of the SPAWNed task's name are "$xx", where xx is a numeric value between 00 and 99. The last 3 characters are the same last three characters of the SPAWNer task.

- Execution of the SPAWNed task begins at the instruction after the call to REX SPAWN if the load module of the SPAWNed task is a duplicate of the load module of the SPAWNer task. If the SPAWNed task's load module was loaded as an overlay from disc, execution starts at the entry point to the load module.

- If the SPAWNer task has I/O in progress when a SPAWN is issued, the SPAWNee may hang in I/O wait forever. Care must be taken to ensure all I/O is completed before SPAWNing a task.

- Resident Tasks may not SPAWN other tasks.

USAGE EXAMPLES

Cause a duplicate of the current task to be created.

```
LDI,R8        SPAWN
LDI,R13       priority (-1 means same as SPAWNer,
REX,MAXIV
```

The new task's name will be returned in R14-R15 If not all resources can be duplicated, a condition code (CC) will be returned to the caller, but the SPAWNed task will be created. It will also get a CC indicating that all resources were not duplicated. Its starting PR will be the same as the SPAWNer's (that is, the instruction following the REX call), but it will get a CC indicating that it is the SPAWNed task.

The following example will perform the same function as the example above, except that if any resource cannot be duplicated, the SPAWNing will not occur. The SPAWNer will receive a CC indicating that the SPAWN failed.

```
LDI,R8        SPAWN+AOE
LDI,R13       priority
REX,MAXIV
```

The following example will SPAWN a task with the same resources as the SPAWNer, but whose body is the overlay specified. This REX uses the same file naming conventions as REX LOVER so a value of 0 means use the same file from which the SPAWNer was loaded and -1 means use the last file that the SPAWNer used in an ACTIVATE, LOVER, or SPAWN call. The referenced file must be a global file. The new task's starting PR is its cataloged entry point. All registers are set as if the task had simply been activated by REX ACTIVATE. The CCs will be set as for the modes above. The AOE option can also be used in this format.

```
LDI,R8        SPAWN+LDFILE
LDI,R12       LFN of the file containing the load module
LDI,R13       priority
LDI,R14       name of the overlay (chars 1-3
LDI,R15       name of the overlay (chars 4-6
REX,MAXIV
```

SET OR GET PORT INFORMATION

Call Name: PORTINFO Call Number: $0122 (REX,MAXIV)

Option Names: FINDPORT Option Values: $0000
GETJID $0001
SETJID $0002
SETTASK $0003
SETSTATUS $0005
GETSTATUS $0006
DEVICES $0007

SERVICE PERFORMED

This service allows the user to inquire about user's logged onto a system with User IDs in the Terminal Control List (TCL).

CALLING PARAMETERS

R8: Call number and option bits
Bits 0-6: =0
Bit 7-15: = Service call number = $122 and
call name = PORTINFO.

R9: Options as described below

R11-R15: See individual options

INDIVIDUAL OPTIONS IN R9

FINDPORT Option

Search the Terminal Control List (TCL) for a match on user ID Return the port number on the nth (R11) occurrence of the User ID. If a user can only log onto one port at a time, n equals 1. If a user can log onto multiple ports concurrently, n is greater than or equal to 1.

Input:

R9: = FINDPORT = $0000
R11 nth occurrence of the User ID to be found.
R13-R15: user ID to be found (CAN-code)

Output:

R12: Port number that the nth occurrence of User ID
is logged onto

Condition Codes Upon Return

NZOC	Condition
1000	R12 contains the port number
0100	nth occurrence of User ID not found, R12 contains zero
0010	R11 contained an invalid count (<= zero)

Programming Considerations:

R11 is used if the User ID may be logged onto more than one port at any given time. The port search always starts with port one. If R11=1, the first matching port is returned. If R11=2, the second matching port is returned. If there is no such entry, a zero and bad CC are returned

```
LDI,R8      PORTINFO
LDI,R9      FINDPORT
LDI,R11     1                 FIND FIRST OCCURRENCE
LDI,R13     @USE
LDI,R14     @R00
LDI,R15     @010
REX,MAXIV
```

GETUID Option

Get the User ID logged onto a specified port. If no port number is specified R12=0 , the port number is determined from the last three characters of the calling task's name.

Input:

R9:	= GETUID = $0001
R12:	Port number (0=my port)

Output:

R12:	Contains port number
R13-R15:	User ID

Condition Codes Upon Return

NZOC	Condition
1000	R12 contains the port number if initially zero, and R13-R15 contains the User ID
0100	No user logged on, R12 contains port number if R12 was initially zero
0010	Port number invalid

Programming Considerations:

If R12 is zero, the port number is determined from the last half of the task name.

```
LDI,R8      PORTINFO
LDI,R9      GETUID
```

```
ZRR,R12                        GET MY USER ID
REX,MAXIV
```

SETUID Option

Store the given User ID in the TCL entry for the specified port. If no port is specified (R12=0), the port number is determined from the last three characters of the calling task's name. The calling task must be privileged to use this option.

Input:

```
R9:         = SETUID = #0002
R12:          Port number (0 = my port)
R13-R15:      User ID to be set for this port
```

Output:

```
R12:          Port number
```

Condition Codes Upon Return

NZOC	Condition
1000	User ID set
0010	Port number invalid
0001	Task not privileged

Programming Considerations.

This option to the PORTINFO REX must be called from a privileged task.

```
LDI,R8       PORTINFO
LDI,R9       SETUID
LDI,R13      @USE
LDI,R14      @R00
LDI,R15      @010
REX,MAXIV
```

SETTASK Option

Set the attentive task name in the TCL entry for my port to the name formed by the concatenation of the first three characters specified and the CAN-code equivalent of my port number.

Input:

```
R9:      = SETTASK = #0003
R15:       First 3 characters of task name in CAN-code
```

Output:

```
R12:       Port number User ID is logged onto.
```

Condition Codes Upon Return

NZOC	Condition
1000	Task name set, and R12 contains the port number
0010	User port number invalid

Programming Considerations:

The port number is determined from the last half of the task name.

```
        LDI,R8      PORTINFO
        LDI,R9      SETTASK
        LDI,R15     @BAT         SET FIRST HALF OF NAME TO "BAT"
        REX,MAXIV
```

SETSTATUS Option

Set status bits in a TCL entry.

Input:

R9:	= SETSTATUS = #0005
R12:	Port number (0 implies my port)
R14:	Mask of desired flags
R15:	New value for flags

Output.

R12:	Port number if initially zero.

Condition Codes Upon Return

NZOC	Condition
1000	Requested flags set and R12 contains port number if initially zero
1100	Same as '1000 but port not logged on
0010	Invalid port
0001	Privilege violation

Programming Considerations:

For the user's own port, any information can be modified; however, for any other port, only the mail waiting flag can be set (it can not be reset). If the calling task is privileged, any flag can be modified. The flag status word is located in the the TCL at word TCLMST.

```
        LDI,R8      PORTINFO
        LDI,R9      SETSTATUS
        ZRR,R12
        LDI,R14     #0002           SET B14
        LDI,R15     #0002           SET IT ON
        REX,MAXIV
```

GETSTATUS Option

Get status bits in a TCL entry.

Input:

```
R9.       = GETSTATUS = $0006
R12.        Port number (0 implies my port)
R14.        Mask of desired flags
```

Output:

```
R12:        Port number if initially zero.
R15:        Requested values for flags.
```

Condition Codes Upon Return

NZOC	Condition
1000	Requested flags in R15 and R12 contains port number if initially zero
1100	Same as '1000 but port not logged on
0010	Invalid port
0001	Privilege violation

Programming Considerations:

For the user's own port, any information can be obtained; however, for any other port, only the message receipt flag is obtainable If the calling task is privileged, any flag can be obtained. The flag status word is located in the TCL at Word TCLMST.

```
LDI,R8       PORTINFO
LDI,R9       GETSTATUS
ZRR,R12
LDI,R14      $0001     GET MAIL WAITING BIT OF MY STATUS WORD
REX,MAXIV
```

DEVICES Option

Get the device names for the input and output channels of the specified port.

Input.

```
R9:       = DEVICES = $0007
R12:        Port number (0 implies my port)
```

Output:

```
R12:        Port number if initially zero
R14:        Device name for the input channel
R15:        Device name for the output channel
```

Condition Codes Upon Return

NZOC   Condition

NZOC	Condition
1000	Requested information returned in R14-R15.
0010	Invalid port

Programming Considerations:

The port number is determined from the last half of the task name.

```
        LDI,R8      PORTINFO
        LDI,R9      DEVICES
        REX,MAKIO
```

TASKWAIT REX

Call Name	TASKWAIT	Call Number:	#0222 (REX,MAXIV
Option Names:	ANYTASK	Option Values:	#0000
	ALLTASK		#0001
	ANYSON		#0002
	ALLSON		#0003

SERVICE PERFORMED

This service allows a task to wait until any (all) task(s) in a list (that the task has spawned) has exited/aborted.

If a task exits and has spawned tasks that are still in the CPUQ, they will be aborted.

CALLING PARAMETERS

R0: Call number:
Bits 0-5 =0
Bits 6-7. ='10 indicates TASKWAIT option within Programmer's Workbench REXs
Bits 8-15: Service call number = $22 and call name = TASKWAIT.

R9: Options as specified for PORTINFO

R10: Address of list of tasks to wait on. The list of task names (double CAN-coded task names) is terminated by a double word zero. (Only if B14 of R9 is reset, that is, option 0 and 1)

CONDITION CODES UPON RETURN

NZOC	Condition
1000	Task wait completed, information returned
0100	No requested task in CPUQ, suspend did not occur
0010	Another task is waiting on a requested task
0011	WORKBENCH not SYSGENed or task not TOCed WORKBENCH

REGISTERS RETURNED

R10-11 Name of the exiting task whose exit caused my resume

R12-15 Completion status of the exiting task

If the exiting task specified that it is returning information to the waiting task (see REX EX T Bit 2), R12-15 will contain whatever information the exiting task passed to the waiting task.

If the exiting task returned no information to the waiting task see REX EXIT Bit 2), R12 will contain a zero. R13 15 will be insignificant.

If the exiting task aborted, R12 will contain a -1, R13-15 will contain the WHO/WHAT/WHERE codes from the abort message.

PROGRAMMING CONSIDERATIONS

With the "ANY" part of the option, only the task name and return codes from the first task to exit are returned to the user. With the "ALL" part of the option, only the task name and return codes from the last task to exit are returned to the user.

INDIVIDUAL OPTIONS IN R9

Wait on ANY Task to Exit

```
LDI,R8          TASKWAIT
LDI,R9          ANYTASK
LDI,R10         Address of a list of task names
REX,MAXIV
```

The first task in the list that exits causes the waiting task to resume with the name of the exiting task returned in R10-11, and the completion status returned in R12-15. If none of the tasks are in the CPUQ, the service returns immediately with a bad condition code. If any of the tasks are being waited on by another task already, the service returns with a bad condition code. If some, but not all of the tasks in the list are still in the CPUQ, the suspend takes place and any status from the already exited tasks is lost.

Wait on ALL Tasks to Exit

```
LDI,R8          TASKWAIT
LDI,R9          ALLTASK
LDI,R10         Address of a list of task names
REX,MAXIV
```

The waiting task is resumed after all tasks in the list have exited the system. Only the name and status from the last task to exit are returned.

Wait on ANY Son to Exit

```
LDI,R8          TASKWAIT
LDI,R9          ANYSON
REX,MAXIV
```

The first task to exit that was spawned by the waiting task causes the waiting task to be resumed.

Wait on ALL Sons to Exit

```
        LDI,R8          TASKWAIT
        LDI,R9          ALLSON
        REX,MAXIV
```

The waiting task is resumed after all of the tasks it spawned have exited the system. Only the name and status of the last task to exit are returned

2.9 EXCLUSIVE USE SERVICES (#23-#24)

TAKE

TAKE EXCLUSIVE USE OF I/O DEVICE

Call Name:	TAKE	Call Number:	#23 (REX,MAXIV)
Option Names:	QUICK	Option Values:	#8000
	PACKTRAN		#4000
	IMMEDIATELY		#2000

SERVICE PERFORMED

This service takes exclusive use of the I/O device assigned to specified logical file. A privileged task can take exclusive use of an entire pack transport if the device is a disc device.

CALLING PARAMETERS

R2: UFT Address

R8: Call number and option bits:
Bit 0: Return from call mode:
=0 Wait for operation to complete.
=1 Quick Return (Return when operation queued). Option name = QUICK.
Bit 1: Type of Take:
=0 Disc, take only assigned device.
=1 Disc, take entire pack transport. Option name = PACKTRAN.
Bit 2: Timing:
=0 Request queued.
=1 Request processed immediately Option name = IMMEDIATELY.
Bits 8-15: Service call number = #23 and call name = TAKE.

[R12:] First 3 characters of CAN-code task name (if not calling task).

[R13:] Second 3 characters of CAN-code task name (if not calling task).

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	- Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

If TAKE immediate is specified, Quick Return has no effect.

R12/13 is used only for privileged tasks to use another task's FAT. These registers are ignored otherwise.

For File Manager files, use the OUS descriptor instead of this service.

USAGE EXAMPLES

Take exclusive use of the line printer. Queue the request but return to me when queued.

```
        LDI,R2        UFT           UFT ADDRESS OF FILE ASSIGNED
*                                   TO PRINTER.
        LDI,R8        TAKE!QUICK    TAKE/QUICK/QUEUED.
        REX,MAXIV
```

I wish to take exclusive use of an entire disc pack immediately.

```
        LDI,R2          UFT         UFT ADDRESS OF FILE ASSIGNED
*                                   TO DISC.
        LDI,R8  TAKE PACKTR IMMEDI
*                                   SELECT SERVICE AND OPTIONS.
        REX,MAXIV                   REQUEST THE SERVICE.
```

GIVE UP EXCLUSIVE USE OF I/O DEVICE

Call Name:	GIVE	Call Number:	$24 (REX,MAXIV)
Option Names:	QUICK PACKTRAN IMMEDIATE	Option Values:	$8000 $4000 $2000

SERVICE PERFORMED

This service gives up exclusive use of the device assigned to specified file. Any operations that have been "sloughed" during the exclusive use period will be immediately requeued to the assigned device.

CALLING PARAMETERS

R2: UFT Address

R8: Call number and option bits:
- Bit 0: Return from call mode:
  - =0 Wait for operation to complete.
  - =1 Quick Return (Return when operation queued). Option name = QUICK
- Bit 1: Type of Take:
  - =0 Disc, take only assigned device.
  - =1 Disc, take entire pack transport. Option name = PACKTRAN.
- Bit 2: Timing:
  - =0 Request queued.
  - =1 Request processed immediately. Option name = IMMEDIATELY.
- Bits 8-15: Service call number = $24 and call name = GIVE.

R12.] First 3 characters of CAN-code task name (if not calling task).

R13:, Second 3 characters of CAN-code task name (if not calling task)

CONDITION CODES UPON RETURN

NZOC	Condition
1000	Normal return (all status bits are in UFT's status word since completion may be asynchronous with respect to the call).

PROGRAMMING CONSIDERATIONS

If a GIVE is issued with the queue option, all previously queued I O requests to the specified device will complete before exclusive use is given.

If GIVE immediate is specified, Quick Return has no effect

R12/R13 is used only for privileged tasks to use another task's File Assign Table (FAT). These registers are ignored otherwise.

This service is not to be used for File Manager files. For File Manager files, use the OJS descriptor instead of this service.

USAGE EXAMPLES

Give up exclusive use of the line printer Queue the request but return to me when queued.

```
        LDI,R2     UFT        UFT ADDRESS OF FILE ASSIGNED TO PRINTER.
        LDI,R8     GIVE!QUICK
*                             GIVE/QUICK/QUEUED.
        REX,MAXIV
```

Give up exclusive use of a disc pack previously taken (immediately).

```
        LDI,R2     UFT        UFT ADDRESS OF FILE ASSIGNED TO ANY
*                             PARTITION ON THE DISC.
        LDI,R8     GIVE!PACKTRANS!IMMEDIATE
*                             GIVE/PACK/NOW.
        REX,MAXIV
```

2.10 ESTABLISH RESIDENCY SERVICES (#27-#28)

ESTABLISH

ESTABLISH THE RESIDENCY OF SPECIFIED NONRESIDENT TASK

Call Name. ESTABLISH   Call Number: #27 (REX,MAXIV)

Option Name. RETTCB   Option Value: #2000

SERVICE PERFORMED

This service makes a nonresident task become loaded and have the characteristics of a resident task. This action will allow the root task program to be patched or examined before execution. The action of establishing a task will stabilize its full TCB in MAP 0 but will not necessarily make the program body unrollable.

CALLING PARAMETERS

R0. Call number and options.
  Bit 2: =0 TCB of established task is not returned.
  =1 TCB of established task is returned. Option name = RETTCB.
  Bits 8-15: Service call number = #27 and call name = ESTABLISH.

[R12:] File name in CAN-code from which the program is to be loaded ("LM" if R12=0).

[R13:] Priority Level (0<P<255, if one not cataloged with task.

R14: CAN-code characters 1-3 of the name of the task to be established. (May be =0 if calling task to be established).

R15: CAN-code characters 4-6 of the name of the task to be established.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0000	Task not queued and not on disc file or, task not queued and MLSEQUENTIAL specified in SYSGEN.
'0001	Tasks have incompatible influence levels.
'0101	TCB too long, pages unavailable in the system.

'0110 Attempt to establish an overlay as a task. (Program was found on the file, but its format was incorrect.)

'0111 No TCB is available for new tasks

REGISTERS CHANGED UPON NORMAL RETURN

R15 = Address of TCB if optioned.

REGISTERS CHANGED UPON ABNORMAL RETURN

R15 = Condition Codes.

Bits 0-11: Zero
Bits 12-15: Same as NZOC

PROGRAMMING CONSIDERATIONS

If the specified task is permanently resident, this service is ignored.

If the task is nonresident but inactive, the allocation of resources and loading of the task is completed but the program is not started. When it is activated, it acts as if it were a permanently resident task except that it is subject to be rolled unless cataloged as unrollable.

If the nonresident task is already active, it is set to the resident and established state.

If the specified task cannot be found or if a task is attempting to establish a task with a higher influence limit, a condition code indicates the error.

USAGE EXAMPLES

Establish the task named "TEST01" which resides on file XYZ.

```
LDI,R12      @XYZ        FILE NAME..
LDI,R13      255         SET PRIORITY LEVEL IF NEEDED
LDI,R14      @TES        SET NAME.
LDI,R15      @T01
LDI,R8       ESTABLISH   CALL NUMBER.
REX,MAXIV                REQUEST THE SERVICE.
HNR,ERROR                HOP IF ERROR.
 -
```

DEESTABLISH

MAKE ESTABLISHED TASK NO LONGER RESIDENT

Call Name: DEESTABLISH Call Number: #28 (REX,MAXIV)

Option Name: RETTCB Option Value: #2000

SERVICE PERFORMED

This service makes an established task no longer have the characteristics of a resident task. If active, no action to remove the task from memory will take place until it exits or is aborted. If the task is not established, this service will perform a NULL operation and set the successful operation condition.

CALLING PARAMETERS

R0: Call number and option bits:
- Bit 2: =0 TCB address of established task is not be returned.
  - =1 TCB address of established task is to be returned. Option name = RETTCB.
- Bits 8-15: Service call number = #28 and call name = DEESTABLISH

R14: CAN-code characters 1-3 of the name of the task to be deestablished. If equal to zero, the calling task is assumed.

R15: CAN-code characters 4-6 of the name of the task to be deestablished.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0000	Task not found.
'0001	Tasks have incompatible influence levels.

REGISTERS CHANGED UPON NORMAL RETURN

R15 = Address of TCB, if optioned.

REGISTERS CHANGED UPON ABNORMAL RETURN

R15 = Condition Codes

Bits 0-11 Zero.
Bits 12-15: Same as NZOC.

PROGRAMMING CONSIDERATIONS

If the specified task is active, it simply becomes no longer "resident" and deallocates its resources only when it exits or is aborted.

If the task cannot be found, or if a task is attempting to deestablish a task with a higher influence limit, a condition code is set to indicate the error.

USAGE EXAMPLE

Deestablish task "TEST01."

```
        LDI,R14         @TES            TASK NAME.
        LDI,R15         @T01
        LDI,R8          DEESTABLISH     CALL NUMBER
        REX,MAXIV                       REQUEST THE SERVICE.
        HNR,ERROR                       HOP IF ERROR.
         -
```

2.11 MEMORY ALLOCATION SERVICES (#29-#2A)

ALLOCATE

ALLOCATE REGION OF PRIVATE MEMORY FOR CALLING TASK

Call Names: ALLOCATE Call Numbers: #29 (REX,MAXIV)

Option Names: IMPATIENT Option Values: #8000
ALLOPERANDS #.00

SERVICE PERFORMED

This service allocates pages of actual memory to those pages of a specified region of task virtual addressing space with which no private or shared actual pages are currently associated and makes all private pages of the specified region accessible to the task for Read,Write usage.

CALLING PARAMETERS

R8: Service Options and Call Number:
 Bit 0: Impatience Specification:
 =0 Wait, if actual pages need to be allocated, but are not currently available. (See PROGRAMMING CONSIDERATIONS.)
 =1 Do not wait, if actual pages need to be allocated, and are not currently available. Option name = IMPATIENT.
 Bit 4: Map Image Specification:
 =0 Map image is task's own IMAP or OMAP.
 =1 Map image is defined by R14 and is in MAP 0 (unprivileged tasks get aborted if Bit 4 is set).
 Bit 6: Active Map Load Inhibit:
 =0 Active map affected is loaded. (Ignored if Bit 4 = 1.
 =1 Active map affected is not loaded.
 Bit 7: Virtual Region Map Reference:
 =0 Virtual region specified exists in the task's instruction map virtual memory.
 =1 Virtual region specified exists in the task's Operand Map virtual memory Option name = ALLOPERANDS
 Bits 8-15: Service call number = #29 and call name = ALLOCATE.

R14]: Map Image Description (if Bit 4 in R8 is set to one and the calling task is resident and is privileged):
 Bits 0-7 Virtual page address in MAP 0(zero) the map image
 Bits 8-15: Length (two's complement, negative representation) of map image.

R15: Virtual Region Description
Bits 0-7: Number of the virtual page (0-255, beginning the region where allocation is to begin.
Bits 8-15: Number of pages in the region (1-256 specified as the low-order eight bits of the two's complement (negative) representation of the positive number of pages in the region

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Region contains only private pages and is accessible for Read/Write usage.
'1001	Region contains at least one shared page. All private pages (if any) of the region are accessible for Read/Write usage.
'0000	The region specified does not lie wholly within the virtual memory defined by the referenced map
'0010	The number of pages which would have to be allocated to satisfy your request is greater than the number ever available for allocation in the system.
'0011	Identical to setting of '0010 except region contains at least one shared page.
'0100	You are impatient and enough pages are not currently available to satisfy your request.
'0101	Identical to a setting of '0100 except region contains at least one shared page.

PROGRAMMING CONSIDERATIONS

When this service is invoked, the definition of the specified region will not be altered in any fashion if a Condition Code Setting other than '100x will be returned

If you supply an Impatience Specification of #0 and actual pages need to be allocated to satisfy your request but are not currently available, this service will take one of the following courses of action.

1 If your system configuration includes the MAX IV Roller task, it will be invoked to "roll out" lower priority tasks until enough pages have been released for allocation to satisfy your request or until your task would be rolled. If your task's request has not been satisfied when further roll-out is impossible, the service will relinquish control to lower priority, unrolled tasks until the required number of pages become available.

2. If your system configuration does not include the MAX IV Roller task, the service will relinquish control to lower priority tasks until the required number of pages becomes available.

Any private pages which are associated with virtual pages of the region when this service is invoked will have their descriptions modified so that these pages are accessible for Read/Write usage.

You may refer to the entire region of virtual memory defined by the referenced map by specifying a region which starts at Page 0 and consists of 256 pages (specified as #0000 in the required notation for input to this service.

The Active Map Load Inhibit option is provided to permit elements of the MAX IV Operating System to use this service. This option is not normally effective when specified by a user task.

The unusual format of the R15 argument results from using the natural MODCOMP IV argument used by the hardware Allocate/Deallocate instructions. This format has been kept compatible even though the hardware Allocate/Deallocate instructions are no longer used (as of MAX IV Revision D . A simple way may be used to express this argument if the MAX IV Macro-Assembler is used.

```
startpage = startword&#FF00
negpage   = -numpages&#FF
R15       = startpage.negpage
```

A task using a map image defined by R14 must not exit or abort while pages are allocated in that map image these pages are unknown to MAX IV and may get lost,.

USAGE EXAMPLE

I want the virtual region of my task's Operand Map virtual memory which starts at Page Number 48 and consists of 16 pages to have private pages accessible for Read/Write usage associated with all non-shared pages in the region.

```
NEGPAG EQU        -16&#FF              GET FUNNY NUMBER FOR
*                                        REGION LENGTH
*      ...
       LDI,R15    48*256!NEGPAG        SET REGION DESCRIPTION.
       LDI,R8     ALLOCATE.ALLOPER     SET MAP REFERENCE AND
*                                        CALL NUMBER.
       REX,MAXIV                       REQUEST THE SERVICE.
       HNR,NOTALO                      HOP IF NOT ALLOCATED.
*      ...
```

DEALLOCATE REGION OF MEMORY FROM CALLING TASK

Call Names: DEALLOCATE   Call Numbers: #2A (REX,MAXIV

Option Names: DEALOPERANDS   Option Values: #80

SERVICE PERFORMED

This service makes all private actual pages in a specified region of my task's virtual memory inaccessible to my task and returns these actual pages to the pool of pages available for allocation to others.

CALLING PARAMETERS

R8: Call number and option bits:

- Bit 4: Map Image Specification and...
  - =0 Map image is task's own IMAP or OMAP.
  - =1 Map image is defined by R14 and is in MAP 0 (unprivileged tasks get aborted if Bit 4 is set).
- Bit 6: Active Map Load Inhibit.
  - =1 Active map affected is not loaded.
  - =0 Active map affected is loaded.
- Bit 7: Virtual Region Map Reference:
  - =0 Specified virtual region exists in the task's Instruction Map virtual memory.
  - =1 Specified virtual region exists in the task's Operand Map virtual memory. Option name = DEALOPERANDS.

- Bits 8-15: Service call number = #2A and call name = DEALLOCATE.

(R14): Map Image Description if Bit 4 in R8 is set and the calling task is resident and privileged:

- Bits 0-7: Virtual page address in MAP 0 of the map image.
- Bits 8-15: Length (two's complement, negative representation) of map image.

R15: Virtual Region Description:

- Bits 0-7: Number of the first virtual page (0-255) in the region to be deallocated.
- Bits 8-15: Number of pages (1-256) in the region specified as the low-order eight bits of the two's complement (negative) representation of the positive number of pages in the region.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Private pages are inaccessible.
'0000	The region specified does not lie wholly within the virtual memory defined by the referenced map.

PROGRAMMING CONSIDERATIONS

When this service is completed successfully, the private pages of the specified region may no longer be referenced without a hardware trap occurring. However, any shared regions which lie wholly or partly within the specified region retain their identity and degree of accessibility which existed prior to invocation of this service.

When this service is invoked, the specified region has not been altered in any fashion if any Condition Code Setting but '1000 is returned.

You may refer to the entire virtual memory defined by the referenced map by specifying a region which starts at Page 0 and consists of 256 pages (specified as #0000 in the required notation for input to this service).

The Active Map Load Inhibit option bit is provided to permit elements of the MAX IV Operating System to use this service. The option is not effective when specified by an unprivileged task.

USAGE EXAMPLE

Deallocate all private pages from the region starting at Page 48 and consisting of 16 pages in my task's Operand Map

```
*          .
        LDI,R15         #30F0           SET REGION DESCRIPTION.
        LDI,R8          DEALLO DEALOP   SET MAP REFERENCE AND CALL.
        REX,MAXIV                       REQUEST THE SERVICE
        HNR,NOTDEA                      XFER IF NOT DEALLOCATED
*       . .
```

EXECUTE LOAD PROGRAM

Call Names:	RXLP	Call Number:	#2B (REX,MAXIV
OPTION NAMES:	RETURN	OPTION VALUES:	#8000
	ABERROR		#4000
	NODIAG		#2000

SERVICE PERFORMED

This service executes a module load program to load a cataloged overlay into task virtual memory.

CALLING PARAMETERS

```
R8:   Call number and option bits.
      Bit 0:    Post-Load Control Transfer Specification:
                =0 Control is transferred to the overlay
                   transfer address after loading of the
                   overlay.
                =1 Control is returned to the invoking
                   task after loading. Option name = RETURN.
      Bit 1:    Abort on Error Specification:
                =0 The invoking task is not aborted if load
                   errors occur and Bit 0 is set to 1.
                =1 The invoking task is aborted if load errors
                   occur and Bit 0 is set to 1.
                   Option name = ABERROR
      Bit 2:    Diagnostic Output Option:
                =0  Diagnostic messages are output when
                    loading errors occur.
                =1  Diagnostic messages are not output
                    when loading errors occur.
                    Option name = NODIAG
      Bit 3:    Get/Use Option:
                =0 The first (or only) segment of the Load
                   Program to be executed lies in the virtual
                   area described by R14 and R15
                =1 The first (or only) segment of the Load
                   Program to be executed resides in the file
                   record whose file Position Index (FPI) is
                   contained in R14 and/or R15.
      Bits 4-5: Ignored.
      Bit 6:    Active Map Load Inhibit Option:
                =0 Active map(s) affected are to be loaded.
                =1 Active map(s) affected are not
                   loaded.
```

Bit 7: =0 Must be 0 (refer to PROGRAMMING CONSIDERATIONS).
Bits 8-15: Service call number = #28 and call name = RXLP.

R12: Module Load Offset (BIAS) specified as a virtual address. If this address is zero, the loader assumes it is loading a non-relocatable module.

R13. Logical File Name (LFN) currently associated with the real file containing the load module and specified in one of the following forms:

- o A 3-character LFN in CAN-code representation.
- o 0 - which is an alternate specification of the LFN from which the Root Task was loaded. A Resident Task uses the LFN specified at System Generation by the TASK statement.
- o =1 - which specifies that the last LFN used by the task for task or overlay loading is to be used.

R14 One or the other of the following items depending on the setting of Bit 3 in R8 (see above):

- o Load Program Directory Entry (Part 1) specified:
  Bits 0-6: Ignored.
  Bit 7: EPI Format Specification:
  =0 if bits 8-15 contain bits 23-16 of the EPI of the first module Load Program Record (LPR).
  =1 if bits 8-15 are to be ignored.
  Bits 8-15: Bits 23-16 of the LPR EPI or Ignored.
- o Task Operand Map Virtual Address of the first word containing the Load Program to be executed.

R15: One or the other of the following items depending on the setting of Bit 3 in R8 (see above).

- o Bits 15-0 of the EPI of the first module LPR.
- o Number of contiguous words starting at the virtual address specified containing the Load Program to be executed.

CONDITION CODES UPON RETURN

NZOC =	Condition
'1000	Load completed successfully and control returned to task after loading. Module entry point in R12. Load module is not relocatable.
'1001	Same as condition code '1000 except load module is relocatable.

'1100	Load completed successfully and control returned to task after loading. No module entry point defined. Load module is not relocatable
'1101	Same as condition code '1100 except load module is relocatable
'0000	No load module of specified name found in file or, no attempt made to load module due to MLSEQUENTIAL specified in SYSGEN.
'0010	Load errors occurred and control returned to task after loading. Module entry point is in R12. Load module is not relocatable.
'0011	Same as condition code '0010 except load module is relocatable.
'0110	Load errors occurred and control returned to task after loading. No module entry point defined. Load module is not relocatable.
'0111	Same as condition codes '0110 except load module is relocatable.
'0101	(Applicable only when Bit 3 of R8 is reset). R15 specified a value not in the range 1-128 inclusive or, in conjunction with the starting Operand Map Virtual Address specified in R15 would have resulted in accessing virtual memory outside the range of the map image or in unallocated memory.

PROGRAMMING CONSIDERATIONS

The maximum number of words that may be used to contain the first words although less may be used. If either a "DEE" Define Module Entry and Exit Loading or "RLP" (Read Load Program Segment) is not encountered before 128 words have been accessed by the Module Loader, a load error - with a diagnostic message if it is not suppressed - will be generated.

A Load Program may be constructed in task virtual memory dynamically by a task as long as all instruction formats are in accordance with those expected by the MAX IV Module Loader

A REX abort will occur if Bit 7 of R8 is set.

The "Programming Considerations" applicable to the REX Load Named Overlay Service are applicable to this service also.

LOAD SPECIFIED OVERLAY PROGRAM

Call Name:	LOVER	Call Number:	#2D (REX,MAXIV)
Option Names:	RETURN	Option Values:	#8000
	ABERROR		#4000
	NODIAG		#2000
	LOOKUP		#1000

SERVICE PERFORMED

This service loads a named cataloged overlay into your task's virtual space from the specified "quick-load" module file.

CALLING PARAMETERS

R8: Call number and option bits:

- Bit 0: Post-Load Control Transfer Specification:
  - =0 Control is transferred to the overlay transfer address after loading of the overlay
  - =1 Control is returned to invoking task after loading. Option name = RETURN.
- Bit 1: Abort on Error Specification:
  - =0 The invoking task is not aborted if load errors occur and Bit 0 is set to one.
  - =1 The invoking task is aborted if load errors occur and Bit 0 is set to 1. Option name = ABERROR
- Bit 2: Diagnostic Output Option:
  - =0 Diagnostic messages are output when loading errors occur.
  - =1 Diagnostic messages are not output when loading errors occur. Option name = NODIAG.
- Bit 3: Determine Presence or Absence of Overlay Option:
  - =0 Load the named overlay as specified per other options in R8.
  - =1 (No loading) Return condition codes indicating presence or absence of load module in specified file.
- Bit 6: Active Map Load Inhibit Option:
  - =0 Active map(s) affected are to be loaded.
  - =1 Active map(s) affected are not loaded
- Bits 8-15: Service call number = #2D and call name = LOVER.

R12 Module Load Offset (BIAS) specified as a virtual address. If this address is zero, the loader assumes it is loading a non-relocatable module.

R13: Logical File Name (LFN) currently associated with the real file containing the load module and specified in one of the following forms

- o A 3-character LFN in CAN-code representation.
- o Ø - which is an alternate specification of the LFN from which the Root Task was loaded. A Resident Task uses the LFN specified at System Generation by the TASK statement.
- o -1 - which specifies that the last LFN used by the task for task or overlay loading is to be used.

R14: Load Module Name (CAN-code characters 1 to 3).
R15: Load Module Name (CAN-code characters 4 to 6).

CONDITION CODES UPON RETURN

NZOC -	Condition
'1ØØØ	If LOOKUP optioned: Load module of specified name found in file. If LOOKUP not optioned: Load completed successfully and control returned to task after loading. Module entry point in R12. Load module is not relocatable.
'1ØØ1	Same as condition code '1ØØØ except load module is relocatable.
'11ØØ	Load completed successfully and control returned to task after loading. No module entry point defined. Load module is not relocatable.
'11Ø1	Same as condition code '11ØØ except load module is relocatable.
'ØØØØ	If LOOKUP optioned. No load module of specified name found in file. If LOOKUP not optioned: No load module of specified name found in file or, no attempt made to load the module due to MLSEQUENTIAL specified in SYSGEN.
'ØØ1Ø	Load errors occurred and control returned to task after loading. Module entry point is in R12. Load module is not relocatable.
'ØØ11	Same as condition code 'ØØ1Ø except load module is relocatable.
'Ø11Ø	Load errors occurred and control returned to task after loading. No module entry point defined. Load module is not relocatable.
'Ø111	Same as condition codes 'Ø11Ø except load module is relocatable.

PROGRAMMING CONSIDERATIONS

If you specify that control is returned after loading and the module contents overlay the location following the "REX,MAXIV" instruction, the results will be unpredictable but confined to your own space.

Any type of load module may be loaded as an overlay

If you specify that control is to be passed to the module after loading and load errors occur, your task will be aborted.

If control is to be passed to the overlay and loading is completed successfully, the contents of R1-R11 and R13-R15 are preserved and passed to the module. The contents of R12 will consist of the virtual address to which control was transferred. The condition code will be set to '1000.

If the LOOKUP option is chosen, the other option bits are ignored and the service returns a condition code of either '1000 (load module of specified name found in file or '0000 (not found in file). No loading takes place, the registers remain unchanged, and R12 (BIAS) is ignored.

USAGE EXAMPLES

Load the overlay named "MYPROG" from the last file I used for loading. I know that the module is edited to operate in my virtual space at an area that is already reserved for it; so I'll specify a Load Offset of zero. Pass control to the module after it is loaded.

```
*       ...
        SUR,R12,R12                       SET LOAD OFFSET.
        GMR,R13,R15                       SET '-1' AS LFN.
        LDMD,R14        OVLNAM            SET OVERLAY NAME
        LDI,R8          LOVER             SET OPTIONS AND CALL
        REX,MAXIV                         REQUEST THE SERVICE.
*       ...
OVLNAM  DFC             @MYP,@ROG         OVERLAY NAME DEFINITION
```

Load the overlay named "MYPROG" from the real file currently associated with the logical file I refer to as "MLM". I know the module is relocatable and I have enough space to load it starting as virtual address 16384, which I will specify as the Load Offset. Return control to me if module loading is successfully completed but abort me if errors occur.

```
*       ...
        LDI,R12      16384      SET LOAD OFFSET.
        LDI,R13      @MLM       SET LFN.
        LDI,R14      @MYP       SET OVERLAY NAME
        LDI,R15      @ROG       ...
        LDI,R8       LOVER.RETURN!ABERROR
*                               SET OPTIONS AND CALL.
        REX,MAXIV               REQUEST THE SERVICE.
        HNR,BADLOD              XFER IF ERRORS OCCURRED
*       ...
```

2.13 TRAP CONTROL SERVICE ($2F

USERTRAP

CAUSE CALLING TASK TO TRAP LOCALLY ON SUBSEQUENT VIOLATIONS

Call Name:	USERTRAP	Call Number:	$2F (REX,MAXIV
Option Names:	USEOFF USEDEBUG	Option Values:	$8000 $4000

SERVICE PERFORMED

A program that violates system rules will normally be aborted. However, if a program calls this REX service it may specify the address of a Local Trap Table (within its own operand addressing space, which contains the addresses of local disposition routines (within its own instruction addressing space) to which the system will transfer control should any of the appropriate violations occur.

CALLING PARAMETERS

R8: Call number and option bits:
 Bit 0: Trap enable bit:
  =0 Local traps are enabled. Subsequent system violations will cause the system to inspect the Trap Table for an appropriate disposition routine to which it will transfer control.
  =1 Local traps are disabled. Subsequent system violations will cause the program to be aborted. Option name = USEOFF.
 Bit 1: For DEBUG processor only.
  =0 Standard application calls.
  =1 For call from DEBUG. Calls of this type require an additional three words in "front" of the standard Trap Table, that is, these words will be addressed as a negative displacement of the address passed in R15 (See "SPECIAL DEBUG USERTRAP" description). Option name = USEDEBUG.
 Bits 8-15 Service call number = $2F and call name = USERTRAP

R15: Trap Table address.
 If this call is being made for the purpose of enabling the USERTRAP (that is, Bit 0 of R8 reset), R15 must contain the address of the Trap Table described subsequently. If this is a USEOFF call Bit 0 of R8 set), R15 need not be loaded with the Trap Table address, although R15 will be changed as described later.

CONDITION CODES UPON RETURN

NZOC   Condition

1000   Normal return.

REGISTERS CHANGED UPON NORMAL RETURN

R15: This register will always be changed to the address of the previous Trap Table. If the USERTRAP service has not been previously called, or if the previous call was USERTRAP!USEOFF, R15 will be set to zero.

REGISTERS CHANGED AT TRAP TIME

None.

SPECIAL DEBUG USERTRAP

A special form of the USERTRAP service and the Trap Table is provided to support specific requirements of the DEBUG processor. When option Bit 1 (USEDEBUG) of R8 is set, the system is informed that the DEBUG processor is the caller and expects an additional 3 words to be added to the front of the standard Trap Table. These words are addressed as a negative displacement of the address loaded into R15.

When used, this call must precede any call by the program being debugged since DEBUG's call will be the only one recorded by the system.

Additionally, the DEBUG processor must either call the USERTRAP!USEDEBUG!USEOFF form of the service, or set the EXIT trap address (word -2) to -1 ($FFFF), prior to exiting the system.

SPECIAL DEBUG TRAP TABLE

The additional 3 words described in the following must precede the standard Trap Table if USEDEBUG is optioned.

WORD   DESCRIPTION

3   USERTRAP and USE request disposition routine address. This word in the DEBUG processor's Trap Table must contain the address of a routine that will simulate subsequent REX USE (MAX III compatible) and REX USERTRAP (MAX IV) requests located in the program being debugged. If the system receives a USE or USERTRAP call while USEDEBUG is active, control is transferred to this address.

NOTE: The system does NOT recognize a -1 in this word.

-2 EXIT disposition routine address or -1
If this word is not -1 (#FFFF) and if the USERTRAP,USEDEBUG is active, any attempt to EXIT will cause the system to transfer control to this address. If this word is -1, EXITs will be permitted.

NOTE: a USERTRAP!USEDEBUG!USEOFF must be issued, or this word must be set to -1, prior to exiting.

-1 REX,#1F (DEBUG) disposition routine address
This word points to the routine that handles REX DEBUG. (See DEBUG description for details.)

NOTE: The system does NOT recognize a -1 in this word.

TRAP TABLE DESCRIPTION

The following describes the standard ^ word trap table required for MAX IV calls. The address of the first word Word 0) is the address that is loaded into R15 prior to an enable call.

WORD	DESCRIPTION
0	Violation address. This word is set by the system, at the time a violation occurs, to the address of the violating instruction. The local disposition routine may examine this word to determine the address of the violating instruction, or the address of the next instruction in the case of a floating point overflow or a transcendental exception. NOTE: DEBUG only: The REX,DEBUG (#1F) disposition routine will find that this word points to the violation address + 1). Floating point and transcendental function disposition routines will find that this word points to the next instruction after the violating instruction.
1	Protect violation disposition routine address or -1 This word must be the address of the disposition routine that will handle protect violations or -1 (#FFFF) if the user wishes the system to abort when a protect violation occurs (normal system response without USERTRAP)
2	Unimplemented REX call disposition routine address or -1. This word must be the address of the disposition routine that will handle unimplemented REX calls or -1 (#FFFF, if the user wishes to abort on unimplemented REX calls.
3	Unimplemented MACRO op-codes(1x, 3x or 5x disposition routine address or -1. This word must be the address of the disposition routine that will handle those op-codes (1x, 3x or 5x) for which hardware implementation or standard software simulation routines have not been supplied, or -1 (#FFFF) if this type of violation should abort.

4 Floating Point overflow disposition routine or -1. This word must be the address of the disposition routine that will handle floating point overflow or underflow violations, or -1 if the user prefers standard system processing of these events. On entry to this routine, condition code O will be set. C will be set if underflow, otherwise reset. This service is used in conjunction with system option 04; if this option is set, then result registers will not be modified. Word 0 points to the next instruction.

5 Transcendental instruction (32/85 only) exception disposition routine or -1. On entry, the condition codes and registers will be as described for the instruction. Word 0 of the trap table will point to the next instruction.

6 Reserved. Must be -1.

MEMORY CELLS CHANGED IF VIOLATIONS OCCUR

If a REX,$2F (MAX III compatible USE call is made and a trap occurs, the address of the violating instruction is stored in absolute address 0 of the calling task's operand addressing space.

USAGE EXAMPLE

Trap all unimplemented REX calls but abort on Protect traps and unimplemented instructions:

```
         LDI,R15   TRAPTB     LOAD TABLE ADDRESS.
         LDI,R8    USERTRAP   USERTRAP CALL.
         REX,MAXIV            ENABLE LOCAL TRAPS.
*        ...
*        ...
*        ...
TRAPTB   RES       1          VIOLATION ADDRESS.
         DEC       -1         ABORT ON PROTECT VIOLATION.
         DEC       UNREX      TRAP ADDRESS FOR UNIMPLEMENTED
*                               REX CALLS.
         DEC       -1         ABORT ON ILLEGAL INSTRUCTIONS.
         RES       3,-1
*        ...
*        ...
UNREX    EQU       $          DISPOSITION ROUTINE.
```

2.14 BYTE STRING ANALYSIS SERVICES (#34-#35)

PARSE NEXT "SIMPLE" PARAMETER IN CHARACTER STRING

Call Name:	GETPAR GET	Call Number:	#34 (REX,MAXIV)
Option Name:	LOWUP	Option Value:	#0000

SERVICE PERFORMED

This service searches through a specified ASCII character string until the end of the next simple parameter is found. It then returns the first (leading) 8 characters of this parameter to the caller. The delimiting character of the parameter is not returned as part of the parameter. If a parameter has less than 8 characters, then trailing blanks are returned to pad out the parameter. The service will optionally convert lowercase alpha characters to uppercase alpha characters.

The search routine will skip leading blanks but will consider any trailing blank, comma, slash, equal sign, or trailing NUL byte to be the end of a valid parameter. If a leading NUL byte is detected before the start of a valid parameter can be found, a special exit will indicate that no parameter was found, thus null delimits the entire parameter string as well as delimiting the last parameter.

CALLING PARAMETERS

R0: Service option and call number:

Bit 0: Alpha case conversion:
=0 DO NOT convert lowercase alpha bytes to uppercase.
=1 Convert lowercase alpha bytes to uppercase. Option name = LOWUP.

Bits 1-7: =0

Bits 8-15: Service call number = #34 and call names are GETPAR and GET.

R10: The base virtual address of the first word of a string to be parsed. This word contains bytes 0 and 1 of the string

R11: The byte displacement, relative to the word address in R10, of the byte in the string where searching of the string will begin. An even displacement selects the left byte of a word. A positive displacement selects a byte higher in address than the base word in R10. Negative displacements are permitted to address bytes whose addresses are lower in memory than the base word

CONDITION CODES UPON NORMAL RETURN

NZOC	Condition
'1000	Normal return, R12 - R15 contains parameter.
'0000	No parameter was found before NUL, R12 - R15 contain blanks.

NOTE: The absence of a parameter (such as ',,,') would return condition codes of 'NZOC = '1000 and R12-R15 could contain all blanks. R12-R15 will contain all blanks when 'NZOC = '0000.

REGISTERS CHANGED UPON NORMAL RETURN

R11: The final byte index will be incremented until it points past (trails) the delimiting character of the parameter. If this service is called again, this will be the next byte examined.

R12: The first 2 characters of the parameter. The right byte may be blank (#20) if the parameter consisted of only one character. The left character will be blank if 2 consecutive nonblank delimiters are found.

R13}
R14} The other characters of the parameter, with the higher numbered registers containing blank characters (#20)
R15} if there are less than 8 characters in the original parameter.

REGISTERS CHANGED UPON ABNORMAL (NO PARAMETER) RETURN

R11: The final byte index. It will point to the NUL byte which terminated the search. If the service is called again, this index will not advance and will always exit abnormal return.

R12}
R13} Dummy parameter, all characters are blank (#20).
R14}
R15}

PARSE NEXT NUMERICAL (OR SIMPLE) PARAMETER IN CHARACTER STRING

Call Name: COLLECT   Call Number: #35 (REX,MAXIV)

Option Name: LOWUP   Option Value: #8000

SERVICE PERFORMED

The COLLECT service searches through a specified ASCII character string until the end of the next numerical parameter or the next simple parameter is found. It then returns the first (leading) 8 characters of a simple parameter, or the first 8 "digit" characters (and decimal point character) of a numerical parameter. The service will optionally convert lowercase alpha characters to uppercase alpha characters.

If a numerical parameter is preceded by one or more "sign" characters, these are suppressed in the returned string but their presence is encoded into unused bits of the string. The delimiting character of the parameter is not returned, and trailing blanks pad the parameter if it has less than 8 characters.

The search routine skips leading blanks but considers any trailing blank, comma, slash, equal sign, trailing "sign" character or trailing NUL byte as the end of a valid parameter. Leading sign characters are skipped but "remembered". Sign characters include the plus sign (+), minus sign (-), dollar sign ($), and hex sign (#). If a leading NUL byte is detected before the start of a valid parameter can be found, a special exit indicates that no parameter was found. Another special exit indicates that a simple, rather than a numerical, parameter was detected.

CALLING PARAMETERS

R8: Service option and call number:

Bit 0: Alpha case conversion:
- =0 DO NOT convert lowercase alpha bytes to uppercase.
- =1 Convert lowercase alpha bytes to uppercase.

Bits 1-7: =0

R8: Bits 8-15: Service call number = #35 and call name = COLLECT.

R10: The base virtual address of the first word of the string to be parsed. This word contains bytes 0 and 1 of the string.

R11: The byte displacement, relative to the word address in R10, of the byte in the string where searching of the string will begin. An even displacement selects the left byte of a word. A positive displacement selects a byte higher in address than the base word in R10. Negative byte displacements are permitted.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return, R12 - R15 contains numerical parameter with leading signs encoded.
'0000	Abnormal return, no parameter was found. R12 - R15 contain all blank characters.
'0001	Simple parameter found instead of numerical parameter. R12 - R15 contains simple parameter.

REGISTERS CHANGED UPON NORMAL (NUMERICAL PARAMETER) RETURN

R11: The final byte index will be incremented until it points past (trails) the delimiting character of the parameter. If this service is called again, this will be the next byte examined. If the delimiter was a trailing "sign", index will point to the sign so that it will be the beginning of the search for the next call.

R12: The first 2 characters of the parameter:

Bit 0:	Negative sign prefix flag: =1 A negative sign (-) preceded parameter.
Bits 1-7:	First ASCII character of parameter (digit or decimal point).
Bit 8:	Hexadecimal sign prefix flag: =1 A hexadecimal sign (#) preceded parameter.
Bits 9-15:	Second ASCII character (digit or decimal point) of parameter, or blank (#20) if parameter consisted of only one character

R13 } The middle 4 bytes of the parameter, or blanks (#20)
R14 } if parameter is short.

R15: The last 2 characters of the parameter:

Bit 0:	Dollar sign prefix flag: =1 Dollar sign ($) preceded parameter.
Bits 1-7:	Seventh character (digit or decimal point) of string or blank (#20).
Bit 8:	String-too-long flag =1 Parameter has more than 8 digits or decimal point characters.
Bits 9-15:	Last character (digit or decimal point) of string or blank (#20).

REGISTERS CHANGED UPON "NON-EXISTENT PARAMETER" RETURN

R11: The final byte index. It will point to the NUL byte which terminated the search. If the service is called again, this index will not advance and will always exit through this abnormal return.

R12 }
R13 } Dummy parameter, all characters are blank (#20).
R14 }
R15 }

REGISTERS CHANGED UPON "SIMPLE PARAMETER" RETURN

R11: The final byte index, same as for normal "numerical parameter" return.

R12: The first 2 characters of a simple parameter. The right byte may be blank (#20) if the parameter consists of only 1 character.

R13 } The remaining characters of the parameter, with higher
R14 } numbered registers containing the trailing blank
R15 } characters if there are less than 8 characters in the parameter.

2.15 CONVERSION SERVICES ($37-$3C)

ATCAN,ATC

CONVERT ASCII STRING TO CAN-CODE

Call Names: ATCAN
ATC

Call Number: $37 (REX,MAXIV,

Option Name: DOUBLE

Option Value: $8000

SERVICE PERFORMED

This service converts any 3-character ASCII string, whose characters belong to the subset of characters that have valid CAN-code weights, into its 16-bit binary CAN code representation. The service optionally converts a 6-character ASCII string into two 16 bit binary CAN-code values. The first three characters form one value and the last three characters form the second value.

CALLING PARAMETERS

R8: Service option and call number:

Bit 0: Double register conversion:
=0 Convert the 3-character ASCII string in R12 and R13
=1 Convert the 6-character ASCII string in R12, R13 and R14. Option name = DOUBLE.

Bits 1-7: =0

Bits 8-15: Service call number = $37 and call names are ATCAN and ATC.

R12: First two ASCII characters of string:
Bits 0-7: First (leading) character of string (byte 0).
Bits 8-15: Second character of string (byte 1).

R13: Third or Third and Fourth characters of ASCII string

o Single Conversion

If Option Bit 0 of R8 is equal to zero:
Bits 0-7: Third (trailing) character of string (byte 2).
Bits 8-15: Ignored byte.

o Double Conversion

If the DOUBLE option is selected, Bit 0 of R8 is set to one.
Bits 0-7: Third character of string (byte 2).
Bits 8-15: Fourth character of string (byte 3).

R14 When the DOUBLE option is selected, R14 is the fifth and sixth characters of the string to convert.

Bits Ø-7: Fifth character of string (byte 4)
Bits 8-15: Sixth trailing character string (byte 5

CONDITION CODES UPON RETURN

NZOC	Condition
*1ØØØ	Normal return
*ØØØØ	Abnormal return. String contained an ASCII character that was not a member of the CAN-code set.

REGISTERS CHANGED UPON NORMAL RETURN

o Single Conversion

If Option Bit Ø of R8 is equal to zero:

R12: The original contents of R13 prior to call.

R13: The 16-bit binary CAN-code representation of the original 3-character string.

o Double Conversion

If the DOUBLE option is selected, Bit Ø of R8 is set to one

R12: The 16-bit binary CAN-code representation of the first three characters (bytes Ø, 1 and 2)

R13. The 16-bit binary CAN-code representation of the last three characters (bytes 3, 4 and 5) specified in the 6-character string.

REGISTERS CHANGED UPON ABNORMAL RETURN

R12: Indefinite

R13: Indefinite.

ATN, ATNUM

CONVERT ENCODED ASCII-STRING TO BINARY NUMBER

CALL NAMES: ATN Call Number: #38 (REX,MAXIV)
ATNUM

SERVICE PERFORMED

This service will convert any encoded ASCII string as is returned by the normal exit of the COLLECT (#35) service into a 16-bit or 32-bit binary integer. The encoded signs will automatically signal either decimal or hexadecimal conversion. Any embedded decimal point character will be removed and returned in the form of a decimal scaling index. The scaling index may be used in conjunction with the binary integer to interpret the position of the original decimal point.

CALLING PARAMETERS

R8: Bits 8-15: Service call number = #38 and call names are ATNUM and ATN.

R12 First 2 (leading) characters of encoded ASCII string:
- Bit 0: Negative flag:
  - =1 Final converted integer two's complemented
  - =0 Final converted integer returned as converted
- Bits 1-7: First character of string and must be a digit (0-9 if decimal, 0-9, A-F if hexadecimal) or a decimal point (.).
- Bit 8. Hexadecimal flag:
  - =1 Hexadecimal conversion algorithm used.
  - =0 Decimal conversion algorithm used.
- Bits 9-15: Second character of string and must be a digit, a decimal point, or a trailing blank if string has only one digit.

R13 } Middle characters of the ASCII string and must be
R14 } digits, decimal point, or trailing blanks.

R15. Last characters of the ASCII string where:
- Bit 0: Dollar sign flag (ignored)
- Bits 1-7: Seventh character of string or blank.
- Bit 8: String-too-long flag:
  - =0 Permits a valid conversion to take place
  - =1 Causes an error exit from this conversion routine.
- Bits 9-15: Last character of string or blank.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return, conversion was successful.
'0000	Abnormal return, original string was longer than 8 characters or more than one decimal point was encountered.

REGISTERS CHANGED UPON NORMAL RETURN

R12: Most significant half of converted binary integer where:
Bit 0: Sign of converted integer if number has more than 15 bits of significance.

R13: Least significant half of converted binary integer where:
Bit 0: Sign of converted integer if R12 contains no significant bits, otherwise, high order bit of 16 bit unsigned numbers.

R14: Flags and scaling factor where:
Bit 0: Significance flag:
=1 if R12 contains any significant bits.
=0 if converted number can be represented in only 16 bits including sign.
Bit 1: Fraction Flag:
=1 if any decimal point was encountered in string.
=0 if no decimal point was encountered in string.
Bits 12-15: Decimal Point scaling index where the binary value indicates the number of significant digits that existed to the right of the original decimal number and =0 if no decimal point was ever present.

NOTE: Since R15 is not changed, the user may test the dollar sign flag ($) in Bit 0 and take appropriate action after conversion of the string

REGISTERS CHANGED UPON ABNORMAL RETURN

R12: }
R13: } Indefinite.
R14: }

CTA, CANTA

CONVERT BINARY CAN-CODE INTO ASCII STRING

Call Names: CTA
CANTA

Call Number: $39 (REX,MAXIV

Option Name: DOUBLE

Option Value: $0000

SERVICE PERFORMED

This service converts any valid 16-bit CAN (compress alphanumeric) code value into the 3-character ASCII string with a trailing blank that it represents. The service optionally converts two valid 16-bit CAN-code values into a 6-character ASCII string.

CALLING PARAMETERS

R8: Service option and call number:

Bit 0: Double register conversion:
=0 Convert one 16-bit CAN-code value in R13.
=1 Convert two 16-bit CAN-code values in R12 and R13. Option name = DOUBLE.

Bits 1-7: =0

Bits 8-15: Service call number = $39 and call names are CTA and CANTA

[R12:] When the DOUBLE option is selected, R12 is the first of two CAN-code values to convert

R13: The binary CAN-code value to be converted. When the DOUBLE option is selected, R13 is the second of two CAN-code values to convert

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return
'0000	Abnormal return, CAN-code has no valid character string representation. Numbers larger than $F9FF in absolute unsigned value are illegal.

REGISTERS CHANGED UPON NORMAL RETURN

o Single Register Conversion

If Option Bit Ø of R8 is equal to zero:

```
R12:  First two characters of string:
      Bits Ø-7:    First character.
      Bits 8-15:   Second character.

R13:  Last character of string.
      Bits Ø-7:    Last character.
      Bits 8-15:   A trailing blank (#2Ø) character.
```

o Double Register Conversion

If the DOUBLE option is selected, Bit Ø of R8 is set one.

```
R12:  First two characters of string:
      Bits Ø-7:    First character
      Bits 8-15:   Second character

R13:  Middle two characters of string:
      Bits Ø-7:    Third character
      Bits 8-15:   Fourth character

R14:  Last two characters of string
      Bits Ø-7:    Fifth character
      Bits 8-15:   Sixth character
```

REGISTERS CHANGED UPON ABNORMAL RETURN

```
R12:  }  Indefinite.
R13:  }

R14:     Indefinite when DOUBLE option is selected.
```

CONVERT SINGLE BINARY NUMBER TO DECIMAL ASCII-STRING

Call Names: BTD
BTDEC

Call Number: #3A (REX,MAXIV)

Option Name: RJUST

Option Value: #8000

SERVICE PERFORMED

This service will convert any 16-bit signed binary integer into a left-justified ASCII string of digits, a leading sign character, and trailing blanks. Optionally, the string will be right-justified with leading blanks replacing unused characters.

CALLING PARAMETERS

R0: Call number and option bits:
Bit 0: Justification flag.
=0 Left-justification with trailing blanks.
=1 Right-justification with leading blanks. Option name = RJUST.
Bits 8-15: Service call number = #3A and call names are BTD and BTDEC.

R13: Signed 16-bit binary integer. If sign bit set, two's complement notation is assumed for negative numbers.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return, string has been converted.

Note: No abnormal return is defined

REGISTERS CHANGED UPON NORMAL RETURN

If left-justification of string is requested:

R11: Leading characters of string:
Bits 0-7: "sign" character of string:
= "-" if sign bit of original binary integer was set(=1)
= "blank" (or #20) if sign bit of original binary integer was reset (=0
Bits 8-15: First digit character of converted string.

```
R12: } The trailing characters of the string and will be
R13: } padded with blanks if string does not have enough
R14: } digits to fill out string.  R14 and R15 will always
R15: } contain blanks.
```

If right-justification of string requested, same as above except:

```
R11: }
R12: } Leading blanks (#20) or leading sign and digits of
R13: } string
R14: }

R15:  Last 2 characters of string:
      Bits 0-7:   Next-to-last digit or sign if string has
                  only 1 digit.
      Bits 8-15:  Least significant digit character of
                  string.
```

HEX, BTHEX

CONVERT BINARY NUMBER TO HEXADECIMAL ASCII-STRING

Call Names: HEX
BTHEX

Call Number: #3B (REX,MAXIV)

Option Name: DOUBLE

Option Value: #0000

SERVICE PERFORMED

This service converts any 16-bit or 32-bit binary value into an ASCII character string in hexadecimal notation. The sign of the number is not interpreted as a sign.

CALLING PARAMETERS

R0: Service option and call number:
Bit 0: Size of binary value
=0 Binary value is 16 bits
=1 Binary value is 32 bits.
Option name = DOUBLE.
Bits 8-15: Service call number = #3B and call names are HEX and BTHEX

R12. If option DOUBLE, R12 is the more significant half of a 32-bit unsigned binary integer.

If not DOUBLE, R12 ignored

R13. If option DOUBLE, R13 is the less significant half of a 32-bit unsigned binary integer.

If not DOUBLE, 16-bit unsigned binary integer.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return.

Note: No abnormal return is currently defined.

REGISTERS CHANGED UPON RETURN

If 16-bit value converted:

R12: The leading bytes of the string (characters 1 and 2):
Bits 0-7: Most significant character of string.
Bits 8-15: Second character of string.

R13: The trailing bytes of the string (characters 3 and 4):
     Bits 0-7. Third character.
     Bits 8-15 Last character of string.

If 32-bit value converted,

R12: Characters 1 and 2 of the string.
R13: Characters 3 and 4 of the string.
R14 Characters 5 and 6 of the string.
R15: Characters 7 and 8 of the string

USAGE EXAMPLE

Convert 32-bit number to 8 hexadecimal characters.

```
*       ..                        CONVERT 32-BIT NUMBER TO 8 HEXADECIMAL
*                                   CHARACTERS.
        LDMD,R12     BIGNUM       LOAD R12 - R13 WITH NUMBER.
        LDI,R8       HEX!DOUBLE   SERVICE CALL + DOUBLE OPTION
        REX,MAXIV                 CONVERT 2ND PART FIRST.
        SFM,R12      BUFFER       RESULT IS R12, R13, R14, R15.
*       ...

BIGNUM  RES 2,0                   STORAGE FOR 32-BIT NUMBER.
BUFFER  RES 4,0                   STORAGE FOR STRING.
```

CONVERT DOUBLE INTEGER TO DECIMAL ASCII STRING WITH DECIMAL POINT INSERTED

Call Names: DTD
DTDEC

Call Number: #3C (REX,MAXIV,

Option Name: RJUST

Option Value: #8000

SERVICE PERFORMED

This service will convert a 32-bit signed number into a sign character and decimal ASCII character string. A decimal point may also be inserted at any specified point in the string. Without a decimal point inserted, the number may be any value that can be expressed as a sign and 9 digits. Insertion of a decimal point will reduce the maximum range to a value that can be expressed with a sign and 8 digits.

CALLING PARAMETERS

R0: Call number and option bits:
- Bit 0: Justification flag:
  - =0 Left-justification with trailing blanks.
  - =1 Right-justification with leading blanks. Option name = RJUST
- Bits 1-7: Unused option bits.
- Bits 8-15: Service call number = #3C and call names are DTD and DTDEC.

R12: Most significant half of binary integer to be converted.
- Bit 0: Sign of number to be converted, and two's complement notation is assumed for negative numbers if set to 1.

R13 Least significant half of binary integer to be converted.

R14: Scaling factor indicating where decimal point should be inserted.

- >8. No decimal point is be inserted
- 0<n<7: Decimal point is inserted so that "n" significant digits show to right of the converted string.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return, conversion was successful.
'0000	Abnormal return, conversion could not fit all digits into fixed length string.

REGISTERS CHANGED UPON NORMAL RETURN

If left-justified string is elected:

```
R11:  Leading characters of string where:
      Bits 0-7:   Sign character (- or blank).
      Bits 8-15:  First digit character of string.

R12: }
R13: } Trailing characters of string and are trailing blanks
R14: } where necessary.
R15: }
```

If right-justified string is elected:

```
R11: }
R12: } Leading characters of string where leading blanks
R13: } are inserted as necessary.
R14: }

R15:  Trailing characters of string where:
      Bits 0-7:   Sign, decimal point, or digit.
      Bits 8-15:  Least significant digit or decimal point.
```

REGISTERS CHANGED UPON ABNORMAL RETURN

```
R11: } All bytes of string contain decimal point (.)
R12: } characters (#2E).  This string is designed to
R13:   represent an overscale value on printed reports.
R14:   Attempted conversion of the value #8000000 yields
R15: } this result.
```

USAGE EXAMPLE

Convert 32-bit binary number to 10 byte decimal ASCII string.

```
*       ...
        LDI,R3         BUFFER       ESTABLISH BUFFER TO STORE STRING.
        LDMD,R12       BIGNUM       LOAD 32 BIT NUMBER TO BE CONVERTED.
        LBR,R14,R11                 MAKE DECIMAL POINT NOT BE INSERTED.
        LDI,R8         DTD+RJUST    RIGHT JUSTIFY STRING.
        REX,MAXIV                   CONVERT  IGNORE ERROR UPON RETURN.
        SFX,R11,R3                  STORE STRING.
*       ...
```

2.16 EVENT LOGGING SERVICE ($3D)

EVELOG

PASS AN EVENT TO THE EVENT LOGGING PACKAGE

Call Name: EVELOG   Call Number: $3D (REX,MAXIV)

Option Name: EVECHO   Option Value: $8000
EVEIMPATIENT $8000

SERVICE PERFORMED

Passes an Event Logging Packet to the Event Logging Facility. The packet will be converted into an Event Logging Node and the data will be copied into the node and sent through the Event Logging process.

CALLING PARAMETERS

R8: Service option and call number:
Bit 0: =0 Relinquish until logging node available
=1 Return with error if logging node not immediately available. Option name = EVECHO and EVEIMPATIENT.
Bits 1-7: =0
Bits 8-15: Service call number = $3D and call name is EVELOG.

R15: Address of the packet to be formed into an Event Logging Node definition in operand addressing space. The format of the packet must be as follows:

Word 0: Logging options and event node type number:

Bit 0: ='0 Event is not echoed
='1 Event is echoed to the system console
Bits 1-7: =0
Bits 8-15: Event node type number.
Refer to "EVENT LOGGING", a chapter in the MAX IV General Operating System, System Guide Manual for information on those event node type numbers reserved for use only by the operating system and those available for the user.

Word 1: Bits 0-7: =0

Bits 8-15: Number of words of event specific information to follow. This number may not exceed the data area size defined for Event Logging Nodes during system generation.

CONDITION CODES UPON RETURN

NZOC	Meaning
'1000	Service performed successfully.
'0100	Service not performed because number of event specific words exceeds configured maximum
'0010	Impatient return was requested and an Event Logging Node was not immediately available.
'0001	Event Logging Facility not in system configuration.
'0011	Event Logging configured but disabled.

USAGE EXAMPLE

Send a user defined event to the Event Logging Facility.

```
        LDI,R0     EVELOG      REX CALL PARAMETER.
        LDI,R15    PKT         ADDRESS OF EVENT PACKET.
        REX,MAXIV              SEND PACKET TO EVENT LOGGING.
*       ...
 PKT    EQU        $           USER DEFINED EVENT PACKET.
        DFC        #0006       DEFINE EVENT NODE TYPE NUMBER.
        DFC        #0004       NUMBER OF WORDS TO FOLLOW.
        DFC        @ABC        FIRST WORD.
        DFC        @DEF        SECOND WORD.
        DFC        3,4         THIRD AND FOURTH WORDS.
```

2.17 ROLL SERVICE ($3E)

ROLL

ROLL A TASK IN/OUT OF MAIN MEMORY

Call Name:	ROLL	Call Number:	$3E (REX,MAXIV)
Option Name:	ROLLIN	Option Value:	$8000
	ROLLOUT		$0000
	ROLLOCK		$4000
	ROLLUNLOCK		$2000
	ROLLNOSUSPEND		$1000

SERVICE PERFORMED

This service can post requests to the Roller task. Requests can be posted to roll a task into main memory or out of main memory. This service can also manipulate the rollability of a task.

If a request to roll a task out of main memory is received, then that task is suspended and a request is posted to the ROLLER task for that task to be rolled out. That task, after being rolled out, will remain rolled out until a roll-in request is posted.

If a request to roll a task into main memory is received, then the task is unsuspended and a request is posted to the ROLLER task for that task to be rolled in. For both roll-in and roll-out requests when the current roll state of the task matches the requested roll state of the task no request will be posted to the ROLLER task.

This service can also make an unrollable task rollable or a rollable task unrollable. Using this capability a task can be cataloged as unrollable due to memory demands and as such will only be rollable, either in or out, through the use of this service.

The task suspend and unsuspend phases of this service are optional

CALLING PARAMETERS

R0: Call number and option bits:

- Bit 0: Roll in or out request.
  - =0 Task roll-out is requested. Option name = ROLLOUT.
  - =1 Task roll-in is requested. Option name = ROLLIN.
- Bit 1: Affect ability of task to roll-out.
  - =0 Ability to roll-out is unaffected
  - =1 Task is marked unrollable. Option name = ROLLOCK.
- Bit 2: Affect ability of task to roll-out:
  - =0 Ability to roll-out is unaffected.
  - =1 Task is marked as rollable. Option name = ROLLUNLOCK

Bit 3: Perform suspend/unsuspend phase of task roll-out/in:
=0 Suspend state of the task is not altered.
=1 Task is not suspended if this is a roll out request or not unsuspended if this is a roll in request. Option name = ROLLNOSUSPEND.

Bits 4-7: =0.

Bits 8-15: Service call number = $3E and call name = ROLL.

R14-R15: The name of the task, in CAN-code, which is to be manipulated.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return.
'1100	Normal return, no request posted to roller task.
'0000	Task not found in CPU queue.
'0001	Incompatible influence limits.
'0010	Roller not configured in system
'0100	Task not rollable because it is resident.

REGISTERS CHANGED UPON NORMAL RETURN

None.

USAGE EXAMPLES

Make task "ABCDEF" rollable, suspend it, and roll it out of main memory until a REX ROLL IN request is posted.

```
        LDMD,R14    TSKNM                          TASK NAME
        LDI,R8      ROLL!ROLLOUT!ROLLUNLOCK        REX CALL
        REX,MAXIV                                  ROLL OUT THE TASK
*       ...
TSKNM   DFC         @ABC,@DEF                      TASK NAME
```

Unsuspend the task "ABCDEF", make it unrollable, and roll it into main memory.

```
        LDMD,R14    TSKNM                          TASK NAME
        LDI,R8      ROLL!ROLLIN!ROLLLOCK           REX CALL
        REX,MAXIV                                  ROLL IN THE TASK
*       ...
TSKNM   DFC         @ABC,@DEF                      TASK NAME
```

2.18 MEMORY DUMP SERVICE ($3F)

DUMP

DUMP REGIONS OF MEMORY IN PRINTED FORMAT

Call Name:	DUMP	Call Number:	$3F (REX,MAXIV)
Option Names:	DECIMAL	Option Values:	$8000
	NOSUPPRESS		$4000
	DINSTRUCT		$2000
	OTHERTASK		$1000
	ACTMEM		$800
	CHARCON		$400
	TOPFORM		$200
	DUXOPT		$100
Extended Options:	DUHEAD		$8000
	PROADR		$4000

SERVICE PERFORMED

This service displays memory regions (actual or virtual) to any specified file. The dump will be printed with either 4, 8, or 16 memory cells per line depending on the carriage width of the printing device and the selected options. Normally, only the virtual operand space or virtual instruction space of the calling task is dumped, but privileged tasks may also dump the virtual addressing spaces of other tasks. In addition, any task may also dump actual memory regions. Any dump lines that contain the same repeated value are suppressed if more than two full repeated lines exist (unless this feature is optioned out).

The dump will appear in either a hexadecimal format or a decimal format. Each line in either the hexadecimal or decimal format may be appended to the right with the ASCII and CAN-code conversions.

If the TOPFORM option is set the dump skips to the top of the next page before dumping.

If the extended option bit, DUXOPT, is set in R8, the following options may be requested:

- The DUHEAD option causes the date and time to be printed at the top of the dump listing.

- The PROADR option permits the user to specify an address to be printed as the start address of the region dumped This value, rather than the address of a memory buffer being dumped, is used to calculate the printed address for each line of output.

CALLING PARAMETERS

```
R0:   Call options and call number:
      Bit 0:  Conversion option:
              =0 Hexadecimal conversion
              =1 Decimal conversion.  Option name = DECIMAL.
      Bit 1:  Suppression control option:
              =0 Suppression of repeated-value lines.
              =1 No suppression of repeated-value lines.
                 Option name = NOSUPPRESS
      Bit 2:  Virtual space selector option:
              =0 Operand space of selected task dumped.
              =1 Instruction space of selected task dumped.
                 Option name = DINSTRUCT.
      Bit 3:  Task selector option:
              =0 Use calling task's virtual space.
              =1 Use virtual space of task whose TCB
                 address is in R1. Option name = OTHERTASK.
      Bit 4:  Mode selector option:
              =0 Virtual memory dumped.
              =1 Actual memory dumped.  Option name =
                 ACTMEM.
      Bit 5:  Character conversion option:
              =0 No character conversion printed.
              =1 Character conversions printed to the
                 right of dump.  Option name = CHARCON.
      Bit 6:  Top of form option
              =0 No top-of-form performed before dump
              =1 A top-of-form performed before dump.
                 Option name = TOPFORM.
      Bit 7:  Extended options:
              =0 No extended options specified.
              =1 Extended options in R9.   Option name =
                 DUXOPT.
      Bits 8-15: Service call number = $3F and call name =
                 DUMP.

R1:   Address of TCB of desired task if Bit 3 of R0 is set to
      one.

R9:   Extended options, if Bit 7 of R0 is set to one:
      Bit 0:  Header option
              = 0 No header will appear
              = 1 The date and time appear at the top of the
                  dump.  Option name = DUHEAD.
      Bit 1:  Print different address:
              = 0 Print actual address of buffer
              = 1 Base address for area to be printed is in R10.
                  Option name = PRDADR

R10:  If Bit 7 of R0 is set  to one and Bit 1 of R9  is set
      to one, the value in R10,  rather than the address of
      the  memory  buffer  being  dumped  will  be  used to
      calculate the address printed for each line of output.

R11:  Base actual page if Bit 4 of R0 is set to one.
```

R12: Logical file name (CAN-code) to which dump is printed.

R13: Dump identification code (CAN-code, to be printed on each line of report.

R14: Starting virtual address of dump (if Bit 4 of R0 reset or zero) or starting word displacement relative to actual page (if Bit 4 of R0 is set to one).

R15: Total number of words to be dumped (0 = 64K).

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return.
'0100	Bad TCB address or non-privileged task attempting to dump another task's space.
'0010	Map size exceeded.
'0001	Task left system or has prologue TCB only.

REGISTERS CHANGED UPON NORMAL RETURN

None

PROGRAMMING CONSIDERATIONS

Any portions of the virtual space being dumped which are unallocated will be flagged in the output with the word UNAL.

USAGE EXAMPLE

Print first 10,000 word region of my program's operand space in decimal on DO file. Then print first 16K region of my instruction space in hexadecimal.

```
*       ...
        LDI,R12      @DO              DIAGNOSTIC OUTPUT FILE.
        LDI,R13      @MYO             "MY OPERANDS" CODE.
        ZRR,R14                       START WITH VIRTUAL ADDRESS 0
        LDI,R15      10000            DUMP 10000 CELLS.
        LDI,R0       DUMP!DECIMA      CONVERT TO DECIMAL.
        REX,MAXIV                     DUMP.
        LDI,R15      #4000            NOW DUMP 16K WORDS AT 0.
        LDI,R13      @MYI             "MY INSTRUCTIONS".
        LDI,R0       DUMP!DINSTR      DUMP INSTRUCTION MAP.
        REX,MAXIV
*       ...
```

2.19 TIME AND SYSTEM STATUS SERVICE ($40)

GETSYS, GETTIME, TIME

GET CURRENT TIME VALUES OR SYSTEM INFORMATION

Call Names:	GETSYS GETTIME TIME	Call Number:	$40 (REX,MAXIV)
Augment Names:	TIMODATE TIMONLY TILLMID MYTIME TASTIM SYSTIM SYSCON USERTIME JULIAN CPUINFO DTSTAMP	Augment Values:	$0000 $0100 $0200 $0300 $0400 $0500 $0600 $0700 $0800 $0900 $0A00

SERVICE PERFORMED

This service will deliver, in general registers and in some cases memory, the current information or time from one of several accumulators in the MAX IV nucleus:

- o The current time-of-day and date in hours, minutes, seconds, ticks, months, days, years, and a ticks-per-second conversion factor.
- o The current time-of-day in minutes and ticks (a positive up count).
- o The elapsed time remaining until midnight (a positive down-count).
- o The CPU time utilized by the calling task or time remaining in limiting task.
- o The total CPU time utilized by all tasks (except IDLE task).
- o The total elapsed time since the system was loaded and started.
- o The configuration of hardware and software of the system.
- o The time (or information) to be returned by a user-coded routine installed at system generation time.
- o JULIAN date in ASCII.
- o System date and time in date/time stamp format.

CALLING PARAMETERS

R8: Call number and call augment codes:
 Bits 4-7: a binary augment code for selection of format:
 = '$0000 Current time-of-day and date are returned in a fully expanded format hours, minutes, seconds, ticks, months, days, years, ticks-per-second. Option name = TIMODATE.

- = '0001 Current time-of-day is returned in a simple minutes and ticks format. Option name = TIMONLY.
- = '0010 Number of minutes and ticks until "midnight" is returned. Option name = TILLMID.
- = '0011 CPU time utilized, or the remaining time until CPU limit expires, is to be returned for the calling task. Option name = MYTIME.
- = '0100 Total time utilized by all tasks (but the idle task) is returned. Option name = TASTIM.
- = '0101 Total time elapsed since the system was loaded and cold-started is returned. Option name = SYSTIM.
- = '0110 Hardware/software environment of the system is returned. Option name = SYSCON.
- = '0111 User-coded time return routine is executed. Refer to J$TIM7 in the User Hooks chapter of the MAX IV Data Structures, SDM Option name = USERTIME.
- = '1000 JULIAN ASCII date is to be returned. Option name = JULIAN.
- = '1001 System CPUID and/or CPUNAME returned. Option name = CPUINFO.
- = '1010 ASCII date/time stamp. Option name = DTSTAMP.
- = '1011 - '1111 are unimplemented and reserved for future use.

Bits 0-15: Service call number = $40 and the call names are GETSYS, GETTIME, and TIME.

R14: If '1001 is chosen for the augment code value, this register contains the address of a buffer in which the CPUNAME is to be returned. The buffer must be in the operand map of the calling task's address space.

If '1010 is chosen, R14 contains the address of a buffer in which the date/time stamp is to be returned This buffer must be at least nine words long and must reside in the task's address space

R15: If '1001' is chosen for the augment code value, this register contains the word size of the buffer that will receive the CPUNAME. If a zero is specified, the CPUNAME will not be returned

If '1010 is chosen, R15 contains the starting byte index into the buffer where the date/time stamp is to be placed. This index is updated to point to the byte position in the buffer after the date/time stamp.

CONDITION CODES UPON RETURN (EXCEPT '.001)

NZOC	Condition
'1000	Normal return - system clock has not been interrupted.
'0000	Abnormal return - system clock has been interrupted by system restart or power failure and the operator has not reset the clock
'0100	Abnormal return - request for unimplemented augment code.

CONDITION CODES ON RETURN (Format '1001)

CC	Value	Meaning
N	1	Normal return, the CPU-ID and the CPU-NAME were both requested or just the CPU-ID was requested and that information was returned; all other CCs will be zero.
N	0	Abnormal return, examine other CC's for reason.
Z	1	CPU-ID was requested but was not initialized.
Z	0	CPU-ID was requested and correctly returned in R13.
O	1	CPU-NAME was requested but was not initialized.
O	0	CPU-NAME was requested and correctly returned in buffer specified.
C	1	Buffer specified was not large enough to hold entire CPU-NAME, but was filled to extent possible.
C	0	CPU-NAME fit into buffer specified.

REGISTERS CHANGED UPON RETURN; FOR "TIMODATE" ('0000) FORMAT

R8. Hours in day (0 to 23).
R9 Minutes in current hour (0 to 59).
R10 Seconds in current minute (0 to 59).
R11: Ticks in current minute 0 to "ticks-per-second"-1).
R12. Months in year (1 to 12).
R13: Days in month (1 to 31). (month dependent limit)
R14: Years (1900 to unlimited).
R15: Conversion factor in "ticks-per-second."

REGISTERS CHANGED UPON RETURN; FOR "TIMONLY" ('0001) FORMAT

R14. Minutes since midnight.
R15: Ticks since last minute count incremented.

REGISTERS CHANGED UPON RETURN; FOR "TILLMID" ('0010) FORMAT

R14: Minutes remaining until midnight.
R15: Ticks remaining in current minute.

REGISTERS CHANGED UPON RETURN; FOR "MYTIME" ('0011) FORMAT

R14: Most significant 16-bits of 32-bit binary time accumulator.
Bit 0: =0 If this time is not a limit, but accumulated CPU time utilized by the calling task.
=1 If this time is a negative CPU time execution limit being incremented toward zero for the calling task.

R15: Least significant part of CPU utilization count or limit:
Bit 15. Least significant bit and represents an accumulation of .005 seconds of time (or one clock interrupt period).

REGISTERS CHANGED UPON RETURN; FOR "TASTIME" ('0100) OR "SYSTIME" ('0101) FORMAT

R14: Most significant 16-bits of a positive 32-bit binary time accumulator
R15. Least significant part (bit 15 increments every clock period which is a fixed .005 seconds).

REGISTERS CHANGED UPON RETURN; FOR "SYSCON" ('0110) FORMAT

R15: Two bytes of hardware/software information respectively where ...
Bit 0: Reserved.
Bit 1: =1 if this is a MAXNET system.
Bit 2: =1 if this is a File Manager system.
Bits 4-8: Reserved.
Bit 9: =1 if host computer is MODCOMP CLASSIC 78xy or CLASSIC II Series.
Bit 10: =1 if host computer is MODCOMP CLASSIC 7830.
Bit 11: =1 if host computer is MODCOMP CLASSIC 32/85.
Bits 12-15. The host operating system (=#4 for MAX IV)

REGISTERS CHANGED UPON RETURN; FOR "JULIAN" ('1000) FORMAT

R13. Last two digits of year in ASCII.
R14: Hundreds and tens digits day in year in ASCII.
R15: Units digit day in year and space in ASCII.

REGISTERS CHANGED UPON RETURN; FOR "CPUINFO" ('1001) FORMAT

R13: The CPU-ID if one exists, otherwise, R13 is not modified

R15: If the CPU-NAME is also requested by a nonzero value in R15 on the call, this register contains the size of the CPU-NAME, in bytes. A zero word is returned in the user buffer immediately following the last word used if the length of the buffer permits. A blank is used to pad the last word used if the CPU-NAME contains an odd number of characters, however, this blank does not count as one of the characters of the CPU-NAME. If no CPU-NAME exists, the buffer is not modified.

REGISTER CHANGED UPON RETURN; FOR "DTSTAMP" ('1010) FORMAT

R15 The byte index is updated to point to the byte position in the buffer following the date/time stamp.

Output from the date/time stamp option

The buffer contains the following 18 characters:

DD MMM YY HH:MM:SS

where

- DD is the day of the month from 1 to 31.
- MMM is the month abbreviation from the set - JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC.
- YY is the year minus 1900.
- HH is the hour from 00 to 23.
- MM is the minute from 00 to 59.
- SS is the second from 00 to 59.

If the system date or time has not been entered, the format is:

00 XXX 00 00:00 00

USAGE EXAMPLE

What time is it and is the clock accurate?

```
        LDI,R8       GETSYS.TIMODATE     CHOOSE FORMAT.
        REX,MAXIV                        REQUEST THE SERVICE.
        HNR,INACCL                       TELL OPERATOR TO SET CLOCK
        SFM,R8       TIMBUF              SAVE TIME IN A BUFFER
*       ...
```

SPECIAL PURPOSE REXS FOR MODCOMP PROCESSORS

Call Name:	INIRES PROCESSOR	Call Number:	#41 (REX,MAXIV)
Option Names:	INIALL INIEMP TOCLOCK PREVENT ALLOW ANLGET DEFMOD	Option Values:	#8000 #4000 #2000 #1000 #0800 #0400 #0200

SERVICE PERFORMED

INIRES initializes either a named or all load module file resident directories to either the loaded or empty state. The primary purpose of this service is for the Task/Overlay Cataloger (TOC) system processor to update a load module file's resident directory following an updating function such as a catalog or compress

TOCLOCK is an interface between TOC and the module loader. It prevents conflicts between module loader and online load module file updates.

ANLGET returns information from memory for the Dump Analyzer (ANL4).

DEFMODE resets job control to its original mode so that it may run under either MAX IV or MAX 32

CALLING PARAMETERS

R8: Service options and call number:

Bits 0-1:	Used by INIRES. See individual description below.
Bits 2-4:	Used by TOCLOCK. See individual description below.
Bit 5:	Used by ANLGET. See individual description below.
Bit 6:	Used by DEFMODE. See individual description below.
Bit 7:	Extended options in R9. Unimplemented.
Bits 8-15:	Service call number = #41. Call name = PROCESSOR.

CONDITION CODES UPON RETURN

See individual descriptions below.

PROGRAMMING CONSIDERATIONS

This service should only be used by MODCOMP PROCESSORS.

INITIALIZE LOAD MODULE FILE RESIDENT DIRECTORY

CALLING PARAMETERS

```
R8.     Service options and call number:
        Bit 0.  Initialize named or all directories:
                = 0 Initialize a single named directory.
                = 1 Initialize all directories configured.
                    Option name = INIALL.
        Bit 1:  Initialize directory to loaded or empty.
                = 0 Initialize directory to loaded state.
                = 1 Initialize directory to empty state
                    Option name = INIEMP.
                    (Cannot be used with the INIALL option).
        Bits 2-7:  = 0
        Bits 8-15: Service call number = #41.
                   Call name = INIRES.
```

[R15] CAN-coded logical file name associated with the specific load module file resident directory if a named directory is to be initialized (R8 Bit 0 = 0).

CONDITION CODES UPON RETURN

NZOC	Condition
*1000	Named directory initialized successfully.
*0010	No load module file resident directory by the specified name is configured.

USAGE EXAMPLES

Initialize the LM Directory to empty.

```
LDI,R8        INIRES!INIEMP     REX CALL PARAMETER
LDI,R15       @LM               DIRECTORY NAME
REX,MAXIV                       INITIALIZE DIRECTORY
```

Initialize the SM directory to loaded

```
LDI,R8        INIRES            REX CALL PARAMETER
LDI,R15       @SM               DIRECTORY NAME
REX,MAXIV                       INITIALIZE DIRECTORY
```

Initialize all directories to loaded

```
LDI,R8        INIRES!INIALL     REX CALL PARAMETER
REX,MAXIV                       INITIALIZE ALL DIRECTORIES
```

LOCK A LOAD MODULE FILE

CALLING PARAMETERS

R8: Service options and call number:

Bits Ø-1:	=Ø	
Bit 2:	=1	Lock or unlock a specified load module or load module file. Option name = TOCLOCK.
Bit 3:		Prevent lock of a load module file:
	=Ø	No lock prevention.
	=1	Prevent the locking of a load module. Called by module loader to inhibit TOC from locking a shared load module while it is loading and to inhibit TOC from locking a segmented program while it is loading or running. Used in conjunction with TOCLOCK (R8 Bit 2 = 1). Option name = PREVENT.
Bit 4.		Allow the locking of a load module.
	=Ø	No release of lock prevention
	=1	Release a previously lock prevented load module so that TOC can manipulate its file. Used in conjunction with TOCLOCK (R8 Bit 2 =1). Option name = ALLOW.
Bits 5-7:	=Ø	
Bits 8-15:		Service call number = $41. Call name = PROCESSOR.

R12: Contains the first half of a CAN-coded module name. If equal to Ø, then the file specified in R15 is locked. If any module had been previously locked on that file, it is unlocked.

R12 cannot be zero when doing a lock PREVENTion or when ALLOWing a previously prevented lock.

R13: Contains the second half of a CAN-coded module name.

R15: Contains a CAN-coded logical file name of a load module file

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	TOCLOCK operation completed successfully.
'Ø1ØØ	No TOCLOCK entry (TOL) available.
'ØØ1Ø	Load module file already locked by another task.
'ØØ11	Invalid load module file name or no TOL in system.
'ØØØ1	Load module locked by another task or load module is lock prevented.
'ØØØØ	Unlock requested by task not having a file locked, or task already has a file locked (only one per task), or ALLOW requested but no module was PREVENTed.

PROGRAMMING CONSIDERATIONS

TOCLOCK is called by TOC to lock and unlock individual load modules and whole load module files. It is a means of communication with the module loader so that the module loader does not try to load a module while TOC is moving that load module in its load module file. Any task running TOC can have one and only one load module file locked at a time; that is, only one TOL entry per task.

TOCLOCK!PREVENT is called by the module loader when it loads a shared load module or segmented program. This prevents TOC from manipulating a multiple-part load module while the module loader is acting upon it. Multiple TOL entries are possible for a TOCLOCK!PREVENT task.

TOCLOCK!ALLOW is called by the module loader when it has completely loaded a shared load module. This makes the shared load module available for TOClocking. Note that the module loader does not do a TOCLOCK.ALLOW once it has loaded a segmented program. A segmented program stays TOCLOCK!PREVENTed until it exits the system. This means that TOC cannot act upon its load module as long as a segmented program it active in the system.

USAGE EXAMPLES

<u>CALL BY TOC</u>

Load a load module file

```
LDI,R8          PROCESSOR!TOCLOCK    LOCK A LOAD MODULE FILE
ZRR,R12                              NO MODULE NAME SPECIFIED
LDI,R15         @TM                  LOAD MODULE FILE NAME
REX,MAXIV
```

Lock a load module

```
LDI,R8          PROCESSOR!TOCLOCK    LOAD A LOAD MODULE
LDI,R12         @ABC                 FIRST HALF OF MODULE NAME
ZRR,R13                              SECOND HALF OF MODULE NAME
LDI,R15         @LM                  LOAD MODULE FILE NAME
REX,MAXIV
```

Unlock a load module

```
LDI,R8          PROCESSOR!TOCLOCK    UNLOCK MY LOAD MODULE
ZRR,R12                              NO MODULE NAME SPECIFIED
LDI,R15         @TM                  LOAD MODULE FILE NAME
REX,MAXIV
```

Unlock a load module file

```
LDI,R8          PROCESSOR TOCLOCK    UNLOCK MY LOAD MODULE FILE
ZRR,R15                              NO FILE NAME SPECIFIED
REX,MAX IV
```

CALLS BY MODULE LOADER

Prevent lock of a load module

```
LDI,R8          PROCESSOR!TOCLOCK.PREVENT
LDI,R12         @ABC                         FIRST HALF OF MODULE NAME
ZRR,R13                                      SECOND HALF OF MODULE NAME
LDI,R15         @LM                          LOAD MODULE FILE
REX,MAXIV
```

Allow lock of a load module

```
LDI,R8          PROCESSOR.TOCLOCK!ALLOW
LDI,R12         @ABC                         FIRST HALF OF MODULE NAME
ZRR,R13                                      SECOND HALF OF MODULE NAME
LDI,R15         @LM                          LOAD MODULE FILE
REX,MAXIV
```

GET INFORMATION FOR DUMP ANALYZER

CALLING PARAMETERS

```
R8     Service options and call number:
       Bits 0-4.  =0
       Bit 5:     =1 Get information for Dump Analyzer (ANL4).
                     Option Name = ANLGET.
       Bits 6-7:  =0
       Bits 8-15: Service call number = #41.
                  Call name = PROCESSOR.

R12:   Memory page displacement.
R13:   Actual memory page number.
R14:   Virtual address of buffer in calling task's omap space.
R15:   Number of words to get from memory.
```

CONDITION CODES UPON RETURN

```
NZOC       Condition

'1000      Successful completion.
'1001      End-of-page reached before all words retrieved.
     R14=  Number of words successfully retrieved from memory
           is returned in R14
'0100      Number of words to retrieve greater than 255 or user
           specified number of words to retrieve as zero.
'0010      Error on store into ANL4 buffer.
     R15=  Remaining number of words yet to be retrieved, user
           must specify next actual page.
```

USAGE EXAMPLE

To get information from memory for the Dump Analyzer ANL4 use the following call:

```
LDI,R8      PROCESSOR!ANLGET      GET INFO FOR ANL4
LDI,R12     X                     MEMORY PAGE DISPLACEMENT
LDI,R13     X                     ACTUAL MEMORY PAGE NUMBER
LDI,R14     BUFFER                VIRTUAL ADDR OF TASK BUFFER
LDI,R15     X                     NUMBER OF WORDS TO GET FROM
                                   MEMORY
REX,MAXIV
```

RETURN JOB CONTROL TO DEFAULT MODE

CALLING PARAMETERS

R8: Service options and call number:

Bits 0-5:	=0
Bit 6:	=1 Return Job Control to default mode. Option name = DEFMODE.
Bit 7:	=0
Bits 8-15:	Service call number = #41. Call name = PROCESSOR.

CONDITION CODES UPON RETURN

NZOC	Condition
1000	Successful completion.

2.21 INTERTASK COMMUNICATIONS SERVICES (#42)

VCOPEN

VIRTUAL CIRCUIT OPEN REQUEST

Call Name:	VCOPEN	Call Number:	#0042 (REX,MAXIV)
Option Names:	ITNOABORT ITNOREPORT ITQUICK	Option Value:	#0001 #0002 #0040

SERVICE PERFORMED

This service sends an open request control message to the task identified by the server named in the calling sequence. The arrival action specified by the server definition is performed.

CALLING PARAMETERS

R0: Call number.
- Bits 0-3: =0
- Bits 4-15: Service call number = #042 and call name = VCOPEN.

R10: Address of server name string in the caller's operand space.

R11: Options:
- Bit 9: Return mode
  - =0 The caller is to await a message from the server before continuing
  - =1 Return immediately after the open message is sent. Option name = ITQUICK.
- Bit 14: Report mode
  - =0 ITC errors are to be reported to Event Logging.
  - =1 ITC errors are not to be reported to Event Logging. Option name = ITNOREPORT.
- Bit 15: Abort mode
  - =0 ITC errors are to abort the calling task.
  - =1 ITC errors are not to abort the calling task. Option name = ITNOABORT.

[R13: Window size in message nodes. If not specified, the SYSGEN default is used. One message node can handle up to 248 words of message.

R14: =0 (Required)

R15: =0 (Required)

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. The operation was performed as requested for a Quick Mode VCOPEN. For a Wait mode operation, the server task replied with a VCACCEPT.

R12: The VC identifier to use in future references to this logical communications path. It is meaningful only on this side of the VC.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3. Status:

- 6 = The server task replied to the "open" request with a VCCLOSE with a reason code. A VCGET or MSGGET may be used to receive the reason code
- -5 = No memory available for a port structure.
- -8 = Calling task has no free VC slot available.
- -9 = Server task has no free VC slot available.
- -16 = Activation failure of server task. Message remains in system.
- -17 = Invalid activation load module file. Message remains in system.
- 20 = Server named in open request not found.
- -41 = Invalid window size (greater than 127 or less than zero).

USAGE EXAMPLE

Send an "open" request to SERVER1 in Quick mode. The calling task will look for a response later using a MSGGET or VCGET, before the first VCSEND is attempted.

```
       LDI,R8         VCOPEN        CALL NUMBER
       LDI,R10        SVNAM         SERVER NAME STRING
       LDI,R11        ITQUICK       QUICK MODE RETURN
       ZRR,R13                      USE THE SYSGEN DEFAULT WINDOW SIZE
       SLRD,R14,R14                 SET TO ZERO FOR VCOPEN
       REX,MAXIV                    REQUEST OPENING OF VC
       RNR,ERROR                    TRANSFER IF NOT SUCESSFUL
       STM,R12        VCNUM         SAVE VC NUMBER
        -
        -
        -
SVNAM  DFC            "SERVER1 "    SERVER NAME STRING
VCNUM  DFC            0             VC IDENTIFIER
```

ACCEPT VIRTUAL CIRCUIT OPEN REQUEST

Call Name:	VCACCEPT	Call Number:	#0142 (REX,MAXIV)
Option Names:	ITNOABORT ITNOREPORT	Option Value:	#0001 #0002

SERVICE PERFORMED

This service confirms the opening of a Virtual Circuit between two tasks by establishing the logical communications path and sending an "accept" control message to the task requesting the VC.

CALLING PARAMETERS

R8: Call number:
- Bits 0-3: =0
- Bits 4-15: Service call number = #142 and call name = VCACCEPT.

R11: Options:
- Bit 14: Return mode
  - =0 ITC errors are reported to Event Logging.
  - =1 ITC errors are not reported to Event Logging. Option name = ITNOREPORT.
- Bit 15: Abort mode
  - =0 ITC errors abort the calling task
  - =1 ITC errors do not abort the calling task. Option name = ITNOABORT.

R12: The VC identifier to use in future references to this logical communications path. It is meaningful only on this side of the VC.

[R13:] Window size in message nodes. If not specified, the SYSGEN default is used. One message node can handle up to 240 words of message.

R14: =0 Required)

R15: =0 (Required)

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

```
R3:      Status:
         0 = Success.  Operation performed as requested.
```

REGISTERS CHANGED UPON ABNORMAL RETURN

```
R3:      Status:
          4   = Invalid window size.  Greater than 127 or
                less than zero.
         -3   = Invalid VC.  The VC does not belong to the
                calling task or does not exist.
```

USAGE EXAMPLE

By using a MSGGET, the calling task had received a VCOPEN request from another task. The calling task wishes to accept the request and establish communications.

```
      LDI,R8        VCACCEPT    CALL NUMBER
      ZRR,R11                   USE REPORT AND ABORT DEFAULTS
      LDM,R12       VCNUM       VC NUMBER FROM PREVIOUS MSGGET
      LDI,R13       20          WINDOW SIZE OF 20 MESSAGE NODES
      SURD,R14,R14              SET TO ZERO FOR VCACCEPT
      REX,MAXIV                 ACCEPT THE VC OPEN REQUEST
       -
      HNR,ERROR
       -
VCNUM DFC           0           VC IDENTIFIER SAVE FROM MSGGET
```

SEND A MESSAGE ON A VIRTUAL CIRCUIT

Call Name:	VCSEND	Call Number:	#0242 (REX,MAXIV)
Option Names:	ITMDATA	Option Value:	#1000
	ITNOABORT		#0001
	ITNOREPORT		#0002
	ITILBYTE		#0004
	ITNOVERIFY		#0100
	ITQDATA		#2000

SERVICE PERFORMED

The VCSEND service sends a message on a specified Virtual Circuit. "M" and "Q" data indicators may be specified in the Options parameter. The contents of the message can reside in discontiguous buffers in the server's space. Up to 32,768 buffer and length pairs can be specified. If the total message length exceeds the space available in the current window, control returns to the caller with status information. Notice that none of the message is sent unless there is sufficient window to send the entire message.

Buffer length is specified in number of words. Messages of odd byte length can be signalled to the receiver with the ignore-last-byte option, ITILBYTE. When this option is used, ITC clears the last byte in the received message to null and makes odd byte status available to the receiver.

CALLING PARAMETERS

R0: Call number:
- Bits 0-3: =0
- Bits 4-15: Service call number = #242 and call name = VCSEND.

R11: Options.
- Bit 2: "Q" data indicator
  - =0 Do not assert "Q" data indicator.
  - =1 Assert "Q" data indicator in received message type. Option name = ITQDATA.
- Bit 3: "M" data indicator
  - =0 Do not assert "M" data indicator.
  - =1 Assert "M" data indicator in received message type. Option name = ITMDATA
- Bit 7: Buffer verification mode
  - =0 Verify buffers for access rights.
  - =1 Do not verify buffers for access rights. Option name = ITNOVERIFY
- Bit 13: Ignore-last byte
  - =0 Do not ignore the last byte. It contains valid data.

=1 Ignore the last byte, because the message is of odd byte length. Option name = ITILBYTE.

Bit 14: Report mode
=0 Report ITC errors to Event Logging.
=1 Do not report ITC errors to Event Logging. Option name = ITNOREPORT.

Bit 15: Abort mode
=0 if ITC errors are to abort the calling task.
=1 if ITC errors are not to abort the calling task. Option name = ITNOABORT

R12: The VC identifier.

R14: Address of a list of buffer and length pairs.

R15: The number of buffer and length pairs.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3. Status:
5 = VC no longer connected. The other side has issued a VCCLOSE.
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.
10 = Protocol violation. The VC is in a state other than "data".
-16 = Activation failure of server task. Message remains in system.
-25 = Invalid buffer address, access rights or length.
26 = Invalid number of segments.
43 = Insufficient window exists to send the entire message. None of the message has been sent.
-44 = Window is closed. None of the message has been sent. A "FLOW CONTROL WINDOW OPEN" message will be received when the window has at least one message node available.

USAGE EXAMPLE

The calling task has already accepted or received an accept on a VC. Now data is to be exchanged on the VC.

```
       LDI,R8          VCSEND          CALL NUMBER
       ZRR,R11                         USE REPORT AND ABORT DEFAULTS
       LDM,R12         VCNUM           VIRTUAL CIRCUIT IDENTIFIER
       LDI,R14         SEGLIS          ADDRESS OF SEGMENT LIST
       LDM,R15         NUMSEG          NUMBER OF SEGMENTS
       REX,MAXIV                       SEND THE MESSAGE

       HNR,ERROR                       ERROR?  IF SO, GO PROCESS
         -
SEGLIS DFC             MSGBF1          ADDRESS OF MESSAGE BUFFER 1
       DFC             LTHBF1          LENGTH OF BUFFER 1 IN WORDS
       DFC             MSGBF2          ADDRESS OF MESSAGE BUFFER 2
       DFC             LTHBF2          LENGTH OF BUFFER 2 IN WORDS

NUMSEG DFC             2               NUMBER OF SEGMENTS IN SEGMENT LIST

VCNUM  DFC             *               VC IDENTIFIER FROM VCOPEN OR MSGGET
```

CLOSE A VIRTUAL CIRCUIT

Call Name: VCCLOSE   Call Number: $0342 (REX,MAXIV)

Option Names: ITNOABORT   Option Value: $0001
ITNOREPORT $0002

SERVICE PERFORMED

This service closes the logical communications path between the two participating tasks. The task issuing the VCCLOSE will not be able to make subsequent accesses to the VC. If the other task continues to send, it will receive a status of "slots not connected", and the messages will be ignored. The VCCLOSE control message may include a 16-bit reason code. Certain codes are reserved. (See PRTDEF R11 options for those codes.)

A VCCLOSE removes all data messages in both directions. This implies that a task willing to do a VCCLOSE or ready to exit without the Save Messages and VCs on Exit option should wait for a reply from the other task indicating that all data has been processed. Otherwise, the data messages that are still in the receive queue are discarded by the VCCLOSE processing. Note that the other task's message remains in its receive queue until the time that it dequeues the "close" control message.

CALLING PARAMETERS

R0: Call number
Bits 0-3: =0
Bits 4-15: Service call number = $342 and call name = VCCLOSE.

R11: Options:
Bit 14: Report mode
=0 if ITC errors are reported to Event Logging.
=1 ITC errors are not reported to Event Logging. Option name = ITNOREPORT.
Bit 15: Abort mode
=0 ITC errors abort the calling task.
=1 ITC errors do not abort the calling task. Option name = ITNOABORT.

R12: The VC identifier.

R13: A 16-bit reason code for the closing of the VC.

R14: =0 (Required)

R15: =0 (Required

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

The calling task desires to terminate communications on a specific VC.

```
       LDI,R8          VCCLOSE      CALL NUMBER
       LDI,R11         ITNOABORT    USE NO ABORT OPTION
       LDM,R12         VCNUM        VC IDENTIFIER
       LDI,R13         #1228        USER SUPPLIED REASON CODE
       SUBD,R14,R14                 CLEAR R14, R15 FOR CALL
       REX,MAXIV                    CLOSE THE VC
        -
       HNR,ERROR                    ERROR? IF SO, GO PROCESS

VCNUM  DFC             0            VC IDENTIFIER FROM VCOPEN OR
*                                   MSGGET
```

RECEIVE A MESSAGE FROM A SPECIFIC VIRTUAL CIRCUIT

Call Name:	VCGET	Call Number:	$0442 (REX,MAXIV)
Option Names:	ITCONTROL	Option Values:	$0010
	ITEXIT		$0060
	ITFLUSH		$0200
	ITQUICK		$0040
	ITNOABORT		$0001
	ITNOREPORT		$0002
	ITILBYTE		$0004
	ITNOVERIFY		$0100
	ITSUSPEND		$0020

SERVICE PERFORMED

The VCGET service returns a message from a specific VC. The message returned will be the oldest data message in the queue, or the first control message if a control message is present.

Whenever the sender selects the ignore-last-byte option (sets ITILBYTE), ITC clears the last byte of the received message to a null. The receiver can request odd byte status by selecting the ITILBYTE option on the VCGET call. When the receiver queries ITC with ITILBYTE Bit 13) set and the sender set ITILBYTE in the option word of the VCSEND, an odd byte status is returned to the receiver in Register 3.

CALLING PARAMETERS

R0: Call number:
- Bits 0-3: =0
- Bits 4-15: Service call number = $442 and call name = VCGET.

R11: Options:
- Bit 6: Flush Message Overflow
  - =0 Save the words in excess of the receive buffer size for the next ITC receive service call.
  - =1 Discard the words in excess of the receive buffer size. Option name = ITFLUSH.
- Bit 7: Buffer Verification mode
  - =0 Verify buffer for access rights
  - =1 Do not verify buffer for access rights. Option name = ITNOVERIFY.
- Bits 9-10: Return mode
  - ='00 Wait for a message to be sent from the other task.
  - ='01 Suspend until a message or other resume occurs. Option name = ITSUSPEND.

='10 Return immediately with or without a message. Option name = ITQUICK.
='11 Exit if no messages are present. Option name = ITEXIT.

Bit 11: Control Message option
=0 Return data and control messages.
=1 Return only control messages. Option name = ITCONTROL.

Bit 13: Ignore-last-byte
=0 Do not indicate odd byte length in R3 status.
=1 Set the appropriate status in R3 if the last byte is not valid data. Option name = ITILBYTE.

Bit 14: Report mode
=0 Report ITC errors to Event Logging.
=1 Do not report ITC errors to Event Logging. Option name = ITNOREPORT.

Bit 15: Abort mode
=0 Abort the calling task for ITC errors.
=1 Do not abort the calling task for ITC errors. Option name = ITNOABORT.

R12: The VC identifier.

R14: The address of the receive buffer.

R15: The length of the receive buffer in words.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status.
0 = Success. Operation performed as requested

R12: The VC identifier corresponding to the received message.

R13: The type of received message.
0 = Data message.
1 = Data message with "M" indicator.
2 = Data message with "Q" indicator
3 = Data message with "M" and "Q" indicators.

The following message types are control message.
4 = Open request. The buffer contains the server name used
5 = Accept acknowledgement. The receive buffer is unchanged.
6 = Close notification. The buffer contains a 16-bit reason code.

12 = Window open on VC. The receive buffer is unchanged.
13 = Reset notification. The buffer contains a 16-bit reason code.

R15: The length of the received message in words.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
7 = Ignore-last-byte option. The receiver should ignore the last byte of the message. This status value is returned only when the VCGET call asks for odd byte information (ITILBYTE, Bit 13 set) and the sender indicated odd byte length by setting ITILBYTE in the option word of the VCSEND.
2 = No message is available in the receive queue.
1 = Message too long for receive buffer. The other registers are as in the normal return. A subsequent ITC receive service call may be used to receive the rest of the message.
-2 = Invalid buffer address, length or access rights.
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.
-29 = Flush option. The message was too long for the receive buffer and the excess message length was discarded. The other registers are as in normal return.

USAGE EXAMPLE

The calling task wants to wait until a message is received from a specific Virtual Circuit.

```
LDI,R8          VCGET           CALL NUMBER
LDI,R11         ITNOABORT!ITNOREPORT
LDM,R12         VCNUM           VC ID FROM A VCOPEN OR MSGGET
LDI,R14         MSGBFR          ADDRESS OF RECEIVE BUFFER
LDI,R15         BFRLTH          LENGTH OF BUFFER IN WORDS
REX,MAXIV                       VCGET, WAIT, NO ABORT OR REPORT
  -
HNR,ERROR                       ERROR? IF SO, GO PROCESS IT
```

MSGGET

GET A MESSAGE FROM ANY VIRTUAL CIRCUIT

Call Name:	MSGGET	Call Number:	#0542 (REX,MAXIV)
Option Names:	ITCONTROL	Option Value:	#0010
	ITEXIT		#0060
	ITFLUSH		#0200
	ITQUICK		#0040
	ITNOABORT		#0001
	ITNOREPORT		#0002
	ITILBYTE		#0004
	ITNOVERIFY		#0100
	ITSUSPEND		#0020

SERVICE PERFORMED

The MSGGET service returns a message from any VC. The message returned will be the oldest data message in the queue, or the first control message if a control message exists in the control message queue.

Whenever the sender selects the ignore-last-byte option (sets ITILBYTE), ITC clears the last byte of the received message to a null. The receiver can request odd byte status by selecting the ITILBYTE option on the MSGGET call. When the receiver queries ITC with ITILBYTE (Bit 13) set and the sender set ITILBYTE in the option word of the VCSEND, an odd byte status is returned to the receiver in Register 3.

CALLING PARAMETERS

R8: Call number:

- Bits 0-3: =0
- Bits 4-15: Service call number = #542 and call name = MSSGET.

R11: Options:

- Bit 6: Flush Message Overflow
  - =0 Save the words in excess of the receive buffer size for the next ITC receive service call.
  - =1 Discard the words in excess of the receive buffer size. Option name = ITFLUSH.
- Bit 7: Buffer Verification mode
  - =0 Verify buffer for access rights.
  - =1 Do not verify buffer for access rights. Option name = ITNOVERIFY.
- Bits 9-10: Return mode
  - ='00 Wait for a message to be sent from the other task.
  - ='01 Suspend until a message or other resume occurs. Option name = ITSUSPEND

=’10 Return immediately with or without a message. Option name = ITQUICK.
=’11 Exit if no messages are present. Option name = ITEXIT.

Bit 11: Control Message option
=0 Return data and control messages.
=1 Return only control messages. Option name = ITCONTROL.

Bit 13: Ignore-last-byte
=0 Do not indicate odd byte length in R3 status.
=1 Set the appropriate status in R3 if the last byte is not valid data. Option name = ITILBYTE.

Bit 14: Report mode
=0 Report ITC errors to Event Logging.
=1 Do not report ITC errors to Event Logging. Option name = ITNOREPORT.

Bit 15: Abort mode
=0 Abort the calling task for ITC errors.
=1 Do not abort the calling task for ITC errors. Option name = ITNOABORT.

R12. The VC identifier.

R14: The address of the receive buffer.

R15: The length of the receive buffer in Words.

CONDITION CODES UPON RETURN

NZOC	Condition
’1000	Normal return. Status in R3 is 0.
’0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3 Status
0 = Success. Operation performed as requested.

R12: The VC identifier corresponding to the received message

R13: The type of received message
0 = Data message.
1 = Data message with "M" indicator.
2 = Data message with "Q" indicator.
3 = Data message with "M" and "Q" indicators.

The following message types are control message:
4 = Open request. The buffer contains the server name used.
5 = Accept acknowledgement. The receive buffer is unchanged.

6 = Close notification. The buffer contains a 16-bit reason code.
12 = Window open on VC. The receive buffer is unchanged.
13 = Reset notification. The buffer contains a 16-bit reason code.

R15: The length of the received message in words.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
7 = Ignore-last-byte option. The receiver should ignore the last byte of the message. This status value is returned only when the MSGGET call asks for odd byte information (ITILBYTE, Bit 13 set) and the sender indicated odd byte length by setting ITILBYTE in the option word of the VCSEND.
2 = No message is available in the receive queue.
1 = Message too long for receive buffer. The other registers are as in the normal return. A subsequent ITC GET service call may be used to receive the rest of the message.
-2 = Invalid buffer address, length or access rights.
-29 = Flush option. The message was too long for the receive buffer and the excess message length was discarded. The other registers are as in the normal return.

USAGE EXAMPLE

The calling task wants to suspend until a message is received, a delay timer resumes the task, an I/O operation completes (resume on I/O complete) or it is resumed by another task or OC. This example assumes that the task was cataloged with a PECULIAR RIOCOMPLETE statement to TOC.

```
         LDI,R8         MSGGET          CALL NUMBER
         LDI,R11        ITNOABORT!ITNOREPORT!ITSUSPEND OPTIONS
         LDI,R14        MSGBFR          ADDRESS OF RECEIVE BUFFER
         LDI,R15        BFRLTH          LENGTH OF BUFFER IN WORDS
         REX,MAKIV                      MSGGET, SUSPEND, NO ABORT OR
*                                       REPORT
         CRI,R3         2               MESSAGE AVAILABLE?
         BZS,NOMSG                      IF NOT, GO CHECK TYPE OF RESUME
         TRR,R3,R3                      DID SOME OTHER ERROR OCCUR?
         BNS,ERROR                      IF SO, GO PROCESS ERROR

GOTMSG EQU              $               PROCESS MESSAGE WITH STATUS =
*                                       0 - 1
```

ENABLE A VIRTUAL CIRCUIT

Call Name: VCENABLE Call Number: #0642 (REX,MAXIV)

SERVICE PERFORMED

This service sets the state of the Enable Status flag for the VC to allow a MSGGET to receive messages from the VC. A VC is in the "enabled" state unless a VCDISABLE has been performed on it (See VCDISABLE). A VC is enabled when first opened.

CALLING PARAMETERS

R8: Call number:
Bits 0-3: =0
Bits 4-15: Service call number = #642 and call name = VCENABLE.

R12: The VC identifier.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

A task has previously "disabled" a Virtual Circuit and now wishes to put it into the "enabled" state.

```
        LDI,R8          VCENABLE        CALL NUMBER
        LDM,12          VCNUM           VC IDENTIFIER FROM A VCOPEN
*                                       OR MSGGET
        REX,MAXIV                       ENABLE THE VC
        BNR,ERROR                       IF ERRORS, GO PROCESS THEM
```

DISABLE A VIRTUAL CIRCUIT

Call Name: VCDISABLE   Call Number: #0742 (REX,MAXIV)

[illegible]

This service resets the state of the Enable Status flag for the VC to prevent a MSGGET from receiving messages from the VC. A VC is in the enabled state unless a VCDISABLE has been performed on it. The disabled state keeps a MSGGET from receiving any messages except a CLOSE message from the specified VC. A VCGET receives messages from the specified VC regardless of the state of the Enable Status flag.

CALLING PARAMETERS

R8: Call number:
- Bits 0-3: =0
- Bits 4-15: Service call number = #742 and call name = VCDISABLE.

R12: The VC identifier.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

A task wants to use a MSGGET to receive messages, but wants to prevent messages from being received from a specific VC.

```
      LDI,R8         VCDISABLE      CALL NUMBER
      LDM,R12        VCNUM          VC IDENTIFIER FROM A VCOPEN
*                                   OR MSGGET
      REX,MAXIV                     DISABLE THE VC
      HNR,ERROR                     IF ERRORS, GO PROCESS THEM
```

LINK TWO VIRTUAL CIRCUITS TOGETHER

Call Name: VCLINK Call Number: #0842 (REX,MAXIV)

SERVICE PERFORMED

In the case where task A has VC1 established to task B and VC2 established to task C, a call to VCLINK by task A connects tasks B and C together and eliminates task A from the logical communications path. Tasks B and C continue to reference the same VC identifiers returned to them during the original VCOPEN device. Messages are forwarded in the appropriate directions and maintain their original ordering. Both VC1 and VC2 must be in the data state. The flow control windows will be as defined by tasks B and C. Note that tasks B and C may continue to execute while the VCLINK is in progress so long as they do not reference the VC that is being linked. In that instance, the task relinquishes until the VCLINK is complete.

CALLING PARAMETERS

R0: Call number
Bits 0-3: =0
Bits 4-15: Service call number = #842 and call name = VCLINK.

R12: The VC identifier of one Virtual Circuit.

R13: The VC identifier of the other Virtual Circuit.

CONDITION CODES UPON RETURN

NZOC	Condition
*1000	Normal return. Status in R3 is 0
*0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.
-4 = Attempt to VCLINK a VC of the calling task that points back to the calling task.
-10 = Protocol violation The VC is in a state other than "data".

USAGE EXAMPLE

A task has a VC open to each of two other tasks (A and B). It now wants to link the two tasks together and be removed from the logical communications path.

```
     LDI,R8        VCLINK         CALL NUMBER
     LDM,R12       VCNUM1         VC IDENTIFIER TO TASK B
     LDM,R13       VCNUM2         VC IDENTIFIER TO TASK A
     REX,MAKIV                    LINK THE TWO VCS TOGETHER
     BNR,ERROR                    ERROR?  IF SO, GO PROCESS IT
```

OBTAIN INFORMATION ABOUT A SPECIFIC VIRTUAL CIRCUIT

Call Name: VCINFO Call Number: #0942 (REX,MAXIV)

SERVICE PERFORMED

This service returns the present state, flow control windows and the name of the task at the other side of the Virtual Circuit. Flow windows are in the range of 0 to 127 and may be temporarily negative due to CLOSE, LINK, or FLOW WINDOW OPEN control messages being sent.

CALLING PARAMETERS

R0: Call number:
Bits 0-3 =0
Bits 4-15. Service call number = #942 and call name = VCINFO.

R12: The VC identifier.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

R10. Current receive flow control window.

R11: Current transmit flow control window.

R13: The state of the Virtual Circuit on the calling task's side.
0 = Error
1 = Not in use
2 = Open request has been received
3 = Open request has been sent
4 = Data

R14: CAN-code characters 1-3 of the other task's name.

R15: CAN-code characters 4-6 of the other task's name.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

Obtain information about a Virtual Circuit of the calling task.

```
       LDI,R8        VCINFO       CALL NUMBER
       LDM,R12       VCNUM        VC ID FROM MSGGET OR VCOPEN
       REX,MAXIV                  GET INFO ON THIS VC
       HNR,ERROR                  IF ERROR, GO PROCESS IT

VCNUM  DEC           0            VC IDENTIFIER
```

OBTAIN INFORMATION ABOUT A SERVER DEFINITION

Call Name: SDINFO Call Number: #0A42 (REX,MAXIV

SERVICE PERFORMED

This service returns information concerning a server definition and options. The user may optionally request that the user data extension to the server definition be returned

CALLING PARAMETERS

[R7]: The address of a 4-word user data table or -1.

R8: Call number:
Bits 0-3: =0
Bits 4-15: Service call number = #A42 and call name = SDINFO.

R10: Address of server name string in the caller's operand space. The string must be terminated by a blank or null byte.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status is R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

R9: The maximum count of VCs that the server may have open simultaneously.

R10: The current count of VCs that the server has open.

R11. The ITC options presently configured for the server task.
Bit 0: The message and VC disposition upon server task exits.
=0 Discard messages and close VCs with reason code -1.
=1 Save messages and allow VCs to remain open

Bit 1: The message and VC disposition upon server task aborts
=0 Discard messages and close VCs with reason code -2.
=1 Save messages and allow VCs to remain open.

Bit 3: Server task is to be activated upon message arrival.

R12: The server task's load module logical file name.

R13. The server's activation priority to use if none was specified when the task was cataloged.

R14: CAN-code characters 1-3 of the server task's name.

R15. CAN-code characters 4-6 of the server task's name.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
-35 = Server name not found.

USAGE EXAMPLE

Obtain information concerning "SERVER1", but do not return the optional user data.

```
       LDI,R7         -1          DO NOT RETURN USER DATA
       LDI,R8         SDINFO      CALL NUMBER
       LDI,R10        SERVER      ADDRESS OF SERVER NAME STRING
       REX,MAXIV                  OBTAIN SERVER INFORMATION
       BNR,ERROR                  IF ERROR, GO PROCESS IT

SERVER DFC            "SERVER1 "
```

DEFINE OR REDEFINE A SERVER TASK'S PORT

Call Name:	PRTDEF	Call Number:	$0B42 (REX,MAXIV)
Option Names:	ITCSOE ITCSOA ITCAMA	Option Value:	$0000 $4000 $1000

SERVICE PERFORMED

This service defines or redefines the parameters contained in a server's port. The port is the ITC extension to a task's TCB that is present when the task is using ITC. Note that ports defined during the SYSGEN cannot be redefined. Also, the maximum VCs parameter cannot be changed once it has been defined.

Two types of ports exist within ITC. A server port is created for tasks that are defined to be servers. The other type, a user port, is built implicitly for other tasks when they issue a call to ITC. A user port resides in the system only as long as the associated task, and may not be redefined into a server port.

CALLING PARAMETERS

R8: Call number:
- Bits 0-3: =0
- Bits 4-15: Service call number = $B42 and call name = PRTDEF.

R9: The number of VCs the server may have concurrently open
- =0 To obtain the default of 4 VCs.
- =1 Up to 57 VCs may be specified.
- (To obtain up to 255 VCs, the port must be defined in the SYSGEN.)

R11. The ITC options to be configured for the server task
- Bit 0: The message and VC disposition upon server task exits
  - =0 Discard messages and close VCs with reason code -1.
  - =1 Save messages and allow VCs to remain open. Option name = ITCSOE.
- Bit 1: The message and VC disposition upon server task aborts.
  - =0 Discard messages and close VCs with reason code -2.
  - =1 Save messages and allow VCs to remain open. Option name = ITCSOA.

Bit 3: Server task is to be activated upon message arrival.
=Ø No action
=1 Activate server task upon message arrival. Option name = ITCAMA.

R12: The server task's load module logical file name.

R13: The server's activation priority to use if none was specified when the task was cataloged.

R14: CAN-code characters 1-3 of the server task's name.

R15: CAN-code characters 4-6 of the server task's name.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is Ø.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3. Status:
Ø = Success. Operation performed as requested

REGISTERS CHANGED UPON ABNORMAL RETURN

R3: Status:
3 = Port parameters for an existing port were successfully redefined.
-13 = Invalid maximum VCs parameter.
-36 = Attempt to redefine a user or SYSGEN port.
-37 = Port is presently being redefined by another task

USAGE EXAMPLE

Define a port for a server task.

```
        LFM,R8          PRTCAL      CALL PARAMETERS
        REX,MAXIV                    DEFINE PORT
        BNR,ERROR                    IF ERROR, GO PROCESS IT

PRTCAL  DFC             PRTDEF      R8. CALL NUMBER
        DFC             1Ø          R9: MAX VCS OF 1Ø
        DFC             Ø           R1Ø: NOT USED
        DFC             ITCSOB!ITCSOA!ITCAMA  R11: OPTIONS
        DFC             @BM         R12: LOAD MODULE FILE LOGICAL
*                                   NAME
        DFC             2ØØ         R13. USE PRI=2ØØ IF NONE
*                                   WAS CATALOGED BY TOC
        DFC             @SVR        R14: SERVER TASK NAME
        DFC             @TSK        R15: SERVER TASK NAME
```

CREATE A SERVER DEFINITION

Call Name: SVRDEF Call Number: #0C42 (REX,MAXIV)

SERVICE PERFORMED

This service creates a server definition that is used to map the server name to a task. An optional table of up to four words of user-supplied data may be kept with the definition. Also, any number of server definitions may point to the same task. A server name may have its associated server task name or user data changed, but the definition cannot be removed from the system. Note also that a SYSGEN-defined server definition is considered to be a permanent definition. This is to facilitate MAX IV restart.

CALLING PARAMETERS

[R7:] The address of a 4-word user data table or -1.

R8: Call number.
Bits 0-3: =0
Bits 4-15: Service call number = #C42 and call name = SVRDEF.

R10. Address of server name string in the caller's operand space. The string must be terminated by a blank or null byte.

R14: CAN-code 1-3 characters of the server task's name

R15: CAN-code 4-6 characters of the server task's name.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

```
     R3:      Status:
               4   = Server parameters successfully redefined.
              -30  = Attempt to change a permanent definition.
              -39  = Invalid character found in server name.
```

USAGE EXAMPLE

Define a server and initialize the user data.

```
       LDI,R7        USRDAT      ADDRESS OF DATA FOR THIS
*                                DEFINITION
       LDI,R8        SVRDEF      CALL NUMBER
       LDI,R10       SVRNAM      ADDRESS OF SERVER NAME
       LDI,R14       @SVR        SERVER TASK NAME
       LDI,R15       @TSK
       REX,MAXIV                  CREATE SERVER DEFINITION
       HNR,ERROR                  IF ERROR, GO PROCESS IT

USRDAT DFC           $A000       USER DATA FOR THIS DEFINITION
       DFC           @LM
       DFC           [redacted]
       DFC           -1

SVRNAM DFC           "SERVER1 "
```

ISSUE A RESET CONTROL MESSAGE

Call Name: VCRESET Call Number: #0D42 (REX,MAXIV)

SERVICE PERFORMED

This service issues a "reset" control message on a specific VC. It has no other effect on ITC except to deliver a 16-bit reason code to the control queue of the task on the other side of the VC.

CALLING PARAMETERS

R8: Call number:
Bits 0-3: =0
Bits 4-15: Service call number = #D42 and call name = VCRESET.

R12. The VC identifier.

R13: A 16-bit reason code for issuance.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Status in R3 is 0.
'0000	Abnormal return. Status detail in R3.

REGISTERS CHANGED UPON NORMAL RETURN

R3: Status:
0 = Success. Operation performed as requested.

REGISTERS CHANGED UPON ABNORMAL RETURN

R3. Status:
3 = Invalid VC. The VC does not belong to the calling task or does not exist.

USAGE EXAMPLE

The calling task needs to notify the other task of some application specific condition that has occurred.

```
        LDI,R8         VCRESET     CALL NUMBER
        LDM,R12        VCNUM       VC ID FROM MSGGET OR VCOPEN
        LDI,R13        #B0AB       USER REASON CODE
        REX,MAXIV                  ISSUE RESET
        HNR,ERROR                  IF ERROR, GO PROCESS IT
           .
           :
VCNUM   DFC            0           VC IDENTIFIER
```

2.22 TASK STATUS SERVICE ($43)

GETASK, TINFORM

GET INFORMATION ABOUT SPECIFIED OR CALLING TASK

Call Name:	GETASK TINFORM	Call Number:	$43 (REX,MAXIV
Option Names:	ALLTOMEM INMEMORY REVARIABLE	Option Values:	$8000 $2000 $1000
Augment Name:	ADRUSE RESUSE DETUSE	Augment Value.	$100 $200 $300

SERVICE PERFORMED

This service will return resource usage and configuration information about any task including the calling task. This information is normally contained in the Task Control Block (TCB) of the task and is not normally accessible to the task program except through this service.

Since the volume of information available is large, and is rarely needed altogether, the information is returned piecemeal according to an "Information Class" selection parameter. The information selected is returned in general registers or in the operand space of the calling task in a memory buffer whose address is specified by the caller.

CALLING PARAMETERS

[R1.] Virtual address of the memory buffer in the calling task's operand space where selected information is to be stored (if option Bit 0 or 2 of R0 is set). On return, R1 will point to the last argument stored +1.

R0: Call options and call number:
Bit 0: =1 Causes all possible information from classes '0000 thru '0.00 to be stored sequentially in the memory buffer whose address is in R1. This buffer is in the calling task's operand space and must be equal to or greater than 30 words. Option name = ALLTOMEM.
Bit 2. Selects the medium for returning the selected information class.
=0 Information returned in general registers.
=1 Information to be returned in memory buffer whose address is in R1. This buffer is in the calling task's operand

space and must be equal to or greater
than 13 words. Option name = INMEMORY.

Bit 3: Information class 0000:
=1 Task's "unspecified variable" should be cleared after retrieving its current contents. This option affects only '0000 class. Option name = REVARIABLE.
Information class 0110:
=0 Instruction map is selected.
=1 Operand map is selected.

Bits 4-7: Selects the Information Class:
= '0000 Basic task information.
= '0001 Addressing space usage.
Option name = ADRUSE.
= '0010 Resource utilization.
Option name = RESUSE.
= '0011 Detailed configuration.
Option name = DETUSE.
= '0100 Task CPU utilization time.
= '0101 Last loaded module information.
= '0110 Virtual page information.
= '0111 User supplied class (USINF7).
= '1000 Extended memory control register.
= '1001 Return requested FAT name.
= '1010 Job Class Information.
= '1011 Extension information.

Bits 8-15: Service call number = #43 and call name = GETASK and TINFORM.

R14-R15: The name of the task, in CAN-code, whose information is to be retrieved. If R14 = 0, the calling task's name will be used.

R14.] Information class '1001: R14 contains the FAT index to be used in retrieving a FAT name.

Information class '1010: R14 contains job information.

Bit 0: =0 Return current state of both bits in R14.
=1 Set selected bits to 1.
Bit 14: =1 Selects BAT bit in TCBRES.
Bit 15: =1 Selects THNA bit in TCBOOS.

This service is used to test and set bits in the user's TCB that are related to batch processing and Job Control.

Information class '1011: R14 contains the maximum length of the transfer in words.

[R9. Information class '1011: R9 contains the option selection.

Bit 0: =0 Get information from selected extension.

=1 Put information into selected
extension.

Bits 1-7: Reserved

Bits 8-15: Extension Selection:

=0 Command line extension
=1 Reserved
=2 User extension
=3-127 Reserved

The extension information service is used to move data to and from a selected extension to the TCB In order to use the command line extension the user must TOC the task with the CLE directive.

The following equates are available for use with this option:

```
COMLIN #0B00        USEREX #0002
PUTEXT #8000        CLESIZ #0041
RETEXT #0000        EXTCLE #0078
COMLEX #0000
```

[R12:] Information class '1011: OMAP information address.
[R13: Information class '1011: Offset in selected extension.

[R13: Information class '0110: R13 must contain the virtual page number in the range of 0 to #FF.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return, task requested is active and stable. Requested information returned.
'0000	Task requested was not found in CPU queue. No information returned.
	If option '1001, invalid FAT index. R14 < 1, or R14 > number of entries in FAT, or task has no FAT.
0001	Task requested found, but calling task has incompatible influence limit. Information will not be returned.
	If option '1011, access rights violation error.
'0010	Task is in CPUQ but not active. Information returned may be incomplete
	If option '1011, extension selection error.
'0011	Task is active but in a transitory state. Information returned may be unstable and incomplete.
0100	Minimum allowable high address of memory buffer (R1) is beyond task's highest address in O-map.

REGISTERS CHANGED UPON Return

Basic Task Information ('ØØØØ)

- R9 Task's job states word
- R1Ø: Task's system option word.
- R11: Task's program option word
- R12: Task's unspecified variable name (in CAN-code),
  = Ø if none has been specified since last reset.
- R13: Task's unspecified variable value (16 bit integer).
  =$ØØØØ if no value specified.
- R14: Task's name (first 3 characters in CAN-code). Name replaces R14(=Ø) if calling task specifies itself and does not know its name.
- R15: Task's name (last 3 characters in CAN-code).
- [R1:] Last address +1 where information stored (not changed if memory storage (option Bit Ø or 2) not optioned).
  If memory storage is optioned, only R1 will be modified and the 7 words will be stored in memory.

Addressing Space Usage ('ØØØ1)

- R3: Logical file name from which last module was loaded.
- R4: Lowest address loaded in I-space.
- R5: Highest address loaded in I-space.
- R6: Highest currently allocated address in I-space.
- R7: Maximum accessible address in I-space.
- R9: Entry point of last module loaded (in I-space).
- R1Ø: Name of last module loaded (first 3 characters CAN-code).
- R11: Name of last module loaded (Last 3 characters CAN-code).
- R12 Lowest address loaded in O-space.
- R13: Highest address loaded in O-space.
- R14: Highest currently allocated address in O-space.
- R15: Maximum accessible address in O-space.
- [R1: Last address +1 where information stored (not changed if memory storage (option Bit Ø or 2 not optioned).
  If memory storage is optioned, only R1 will be modified and the 12 words will be stored in memory.

Resource Utilization ('ØØ1Ø)

- R6: Number of I/O nodes (maximum number of concurrent I/O operations possible).
- R7: Number of I/O operations in progress.
- R9: Current size of service buffer (=Ø if not currently allocated).
- R1Ø: Number of private pages currently in use by task.
- R11: Maximum number of page-sharing regions permissible by task.
- R12: Current size of PS/PR stack.
- R13: Words used in PS/PR stack
- R14: Current size of main stack.
- R15: Words used in main stack
- [R1.] Last address +1 where information stored (not changed if memory storage (option Bit Ø or 2) not optioned).
  If memory storage is optioned, only R1 will be modified and the 9 words will be stored in memory.

Detailed Configuration ('0011)

R7: Virtual address (in MAP 0) of task's TCB (prologue Word 0).

R9: Task's importance word:
- Bits 0-7: Task's influence level.
- Bits 8-15: Task's execution priority level.

R10: Task's resource PS word and task type:
- Bits 0-2: Instruction Map (IM) assigned.
- Bit 3: =1 If root task privileged.
- Bits 4-7: Register Block (RB) assigned.
- Bits 8-10: Operand Map (OM) assigned.
- Bit 11: =1 If task is a symbiont.
- Bit 12: =1 If task is a batch-processing task.
- Bit 13: =1 If task is an exceptional conditions task.
- Bit 14: =1 If task is the "OC" task.

R11: Task's characteristics:
- Bit 0. =1 If task is resident or fully established.
- Bit 1. Reserved.
- Bit 2: Reserved.
- Bit 3: =1 If task is established resident or being established.
- Bit 4: =1 If task is remote (has local OC/CO files).
- Bit 5: =1 If task will be started if system started/ restarted
- Bit 6: Reserved.
- Bit 7: Reserved.
- Bit 8: Reserved.
- Bit 9: Reserved.
- Bit 10: =1 If task currently uses only MAP 0 (0)
- Bit 11: Reserved
- Bit 12: Reserved.
- Bit 13: Reserved.
- Bit 14: Reserved.
- Bit 15: Reserved.

R12: Number of pages in task's instruction map addressing space (0=256 pages).

R13: Number of pages in task's operand map addressing space (0=256 pages).

R14: Virtual entry point of root task in instruction space

R15: Logical file name (CAN-code, dedicated for task load modules.

(R15) Last address +1 where information stored (not changed if memory storage (option bits 0 or 2) not optioned). If memory storage is optioned, R7 and R9 through R15 will not be modified and the 8 words will be stored in memory

Task CPU Time Utilization ('0100)

R14: Number of clock interrupt periods (.005 seconds) used since the task was activated (most significant part of 32-bit count).

R15: Least significant part of number of clock interrupt periods - where Bit 15 is incremented for each clock interrupt.

[R1:] Last address +1 where information stored (not changed if memory storage (option Bit 0 or 2) not optioned). If memory storage is optioned, R14 and R15 will not be modified and the 2 words will be stored in memory.

Last Loaded Module Information ('0101)

R3: Logical file name from which last module was loaded.

R4 Lowest address loaded of last module in I-SPACE.

R5. Highest address loaded of last module in I-SPACE.

R6. Highest addressing space (from TOC SPACE/ISPACE) of last loaded module in I-SPACE.

R7: Maximum accessible (MAP size) address in I-SPACE.

R9. Entry point of last module loaded in I-SPACE.

R10: Name of last module loaded (first 3 characters CAN-code)

R11: Name of last module loaded (last 3 characters CAN-code).

R12: Lowest address loaded of last module in O-SPACE.

R13: Highest address loaded of last module in O-SPACE

R14: Highest addressing space (from TOC SPACE/O-SPACE) of last loaded module in O-SPACE.

R15: Maximum accessible (MAP size) address in O-SPACE.

[R1:] Last address +1 where information stored (not changed if memory storage (option Bit 2) not optioned). If memory storage is optioned, only R1 will be modified and the 12 words will be stored in memory.

Virtual Page Information ('0110)

R12: Information returned from MAP image (access rights, shared memory, actual page). This register will be set to zero if no memory is allocated or the wanted virtual page (in R13) is outside the bounds of the MAP.

[R1:] Last address +1 where information was stored (not changed if memory storage (option Bit 2) not optioned). If memory storage is optioned, only R1 will be modified and the one word will be stored in memory.

User Supplied ('0111)

This is a user supplied information service and will be called locked out at level F with the registers set as follows:

R1: TCB address of the task whose information is to be retrieved.
R2: Return address from the user's service.
R3: Destroyed
R4: Destroyed.
R5: Have not been modified.
R6: Have not been modified.
R7: Destroyed.
R8-R15: Have not been modified.

The user supplied information service will be called with "BLM,2 U$INF7". If the user does not supply a "U$INF7" routine and this service class ('0111) is called, a normal return with no error will be executed

Extended Memory Control Register ('1000)

R15: Value of extended memory limits control register for this task.
R1: Last address + 1 where information was stored (not changed if memory storage (option Bit 2) not optioned). If memory storage is optioned, only R1 will be modified and the one word will be stored in memory.

Return Requested FAT Name ('1001)

R15: Contains the FATNAM for the FAT index specified. A value of zero is returned if the requested FAT is vacant.

Job Class Information ('1010)

R14: Contains the states of the BAT bit in TCBRES as B14 and the THNA bit in the TCBOOS as B15.

MODIFY SYSTEM OPTIONS OF SPECIFIED TASK

Call Name: MODOPTION Call Number: #50 (REX,MAXIV)

Option Name: OTASK Option Value: #0000

SERVICE PERFORMED

This service modifies the System Option Word of the Task Control Block (TCB) of requesting task or another task, as specified.

CALLING PARAMETERS

```
R8: Service Option and Call Number:
    Bit 0:   Task Specification:
             =0   Modify the System Option Word of my task.
             =1   Modify the System Option Word of the task
                  whose name is contained in R12-13.
                  Option name = OTASK.
    Bits 8-15:    Service call number = #50 and the call name
                  = MODOPTION.

R12: CAN-code characters 1-3 of the name of the task whose
     System Option Word is to be modified if not calling task.

R13: CAN-code characters 4-6 of the name of the task whose
     System Option Word is to be modified if not calling task

R14  Option Modification Specification that indicates which
     portion(s) of the System Option Word is to be modified
     and is specified as a 16-bit value which contains:
             =0 If the corresponding bit of the System Option
                Word is not to be modified.
             =1 If the corresponding bit of the System Option
                Word is to be modified.

R15: Option Modification Value that indicates the new values
     to replace the contents of the fields indicated by the
     Option Modification Specification in R14.
```

CONDITION CODES UPON RETURN

```
 NZOC   Condition

'1000 - Modification completed successfully.

'0000 - Another task specified and task is not currently
        resident or scheduled in the system.
```

'0001 Another task specified and your influence limit does not permit you to modify the System Option Word of that task.

PROGRAMMING CONSIDERATIONS

If only the System Option Word of the caller's task is being modified, there is no need to test the condition code settings on return.

The operation of this service will preclude the modification of any fields other than those specified in R14, regardless of the contents of R15

USAGE EXAMPLES

Modify the 'Loader Hold' bit of my System Option Word so that loading of my modules does not result in a console message and 'Hold' that is, set Bit 14 of the word to 0; and set the "UO" bit to 1.

```
 0      EQU           0
*       ...
        LDI,R14       U0+HL            SET MOD. SPEC.
        LDI,R15       U0               SET NEW VALUES.
        LDI,R8        MODOPT           SET "MY TASK"/CALL NUMBER.
        REX,MAXIV                      REQUEST THE SERVICE.
*       ...                            DON'T BOTHER TO TEST CC'S.
```

Modify the "UO" bit of the task named "MONAMI" to set bit to '1.

```
U0      EQU           0
*       ...
        LDI,R12       @MON             SET TASK NAME.
        LDI,R13       @AMI              -
        LDI,R14       U0               SET MOD. SPEC.
        TRR,R15,R14                    SET NEW VALUE.
        LDI,R8        MODOPT+OTASK     SET "HIM"/CALL NUMBER.
        REX,MAXIV                      REQUEST THE SERVICE.
        HNR,NOTINS                     XFER IF PROBLEMS.
*       ...
```

GET SYSTEM OPTIONS OF SPECIFIED TASK

Call Name: GETOPT Call Number: #51 (REX,MAXIV)

Option Name: OTASK Option Value: #8000

SERVICE PERFORMED

This service returns the contents of the System Option Word of the Task Control Block (TCB) of the requesting task or another task.

CALLING PARAMETERS

R8: Service Option and Call Number:
- Bit 0: Task Specification:
  - =0 Get the System Option Word of my task.
  - =1 Get the System Option Word of the task whose name is contained in R12-R13. Option name = OTASK.
- Bits 8-15: Service call number = #51 and call name = GETOPT.

R12: CAN-code characters 1-3 of the name of the task whose System Option Word is requested.

R13: CAN-code characters 4-6 of the name of the task whose System Option Word is requested.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0000	Another task was specified and that task is not currently resident or scheduled in the system.
'0001	Another task was specified and your influence limit does not permit you to get the System Option Word of that task.

REGISTERS CHANGED UPON RETURN

R15: The contents of the System Option Word.

PROGRAMMING CONSIDERATIONS

If the System Option Word of the caller's task is referenced, there is no need to test the condition code settings upon return.

USAGE EXAMPLES

Get my System Option Word contents.

```
*       ...
        LDI,R8      GETOPT          SET TASK NAME.
        REX,MAXIV                   REQUEST THE SERVICE.
*       ...                         R15 CONTAINS VALUE NOW.
```

Get the System Option Word contents of the task named 'MONAMI'.

```
*       ...
        LDI,R12     @MON            SET TASK NAME.
        LDI,R13     @AMI            ...
        LDI,R8      GETOPT+OTASK    SET OPTION/CALL NUMBER.
        REX,MAXIV                   REQUEST THE SERVICE.
        HNR,ERROR                   R15 CONTAINS VALUE NOW.
```

MODIFY PROGRAM OPTIONS OF SPECIFIED TASK

Call Name: MODPOP   Call Number: #52   (REX,MAXIV)

Option Name: OTASK   Option Value: #8000

SERVICE PERFORMED

This service modifies the Program Option Word of the Task Control Block (TCB) of the requesting task or another task.

CALLING PARAMETERS

R0: Service Option and Call Number:
- Bit 0: Task Specification:
  - =0 Modify the Program Option Word of my task
  - =1 Modify the Program Option Word of the task whose name is contained in R12-13. Option name = OTASK.
- Bits 8-15: Service call number = #52 and the call name = MODPOP.

R12: CAN-code characters 1-3 of the name of the task whose Program Option Word is requested.

R13: CAN-code characters 4-6 of the name of the task whose Program Option Word is requested.

R14: Option Modification Specification that indicates which portion(s) of the Program Option Word is to be modified and is specified as a 16-bit value which contains:

- =0 If the corresponding bit of the Program Option Word is not to be modified.
- =1 If the corresponding bit of the Program Option Word is to be modified.

R15: Option Modification Value that indicates the new values to replace the contents of the fields indicated by the Option Modification Specification in R14.

CONDITION CODES UPON RETURN

NZOC	Condition
*1000	Modification completed successfully.
*0000	Another task was specified and task is not currently resident or scheduled in the system.

0001 - Another task was specified and your influence limit does not permit you to get the Program Option Word of that task.

PROGRAMMING CONSIDERATIONS

If only the Program Option Word of the caller's task is being modified, there is no need to test the condition code settings on return.

The operation of this service will preclude the modification of any fields other than those specified in R14, regardless of the contents of R15.

USAGE EXAMPLES

Modify Bits 3 and 4 of my Program Option Word to 0 and 1 respectively.

```
C0      EQU          0
*       ...
        LDI,R14      #1100          SET MOD. SPEC.
        LDI,R15      #0.00          SET NEW VALUES.
        LDI,R3       MODPOP         SET "MY TASK"/CALL NUMBER.
        REX,MAXIV                   REQUEST THE SERVICE.
*       ...                         DON'T BOTHER TO TEST CC'S
```

Modify B7 and B15 of the Program Option Word of the task named "MONAMI" to be set.

```
J0      EQU          0
*       ...
        LDI,R12      @MON           SET TASK NAME.
        LDI,R13      @AMI           ...
        LDI,R14      #0.01          SET MOD. SPEC.
        LDI,R15,R14                 SET NEW VALUE.
        LDI,R3       MODPOP+OTASK   SET "HIM"/CALL NUMBER.
        REX,MAXIV                   REQUEST THE SERVICE.
        HNE,NOTINS                  XFER IF PROBLEMS.
*       ...
```

GET PROGRAM OPTIONS OF SPECIFIED TASK

Call Name: GETPOP   Call Number: #53 (REX,MAXIV

Option Name: OTASK   Option Value: #0000

SERVICE PERFORMED

This service returns the contents of the Program Option Word of the requesting task or another task.

CALLING PARAMETERS

R0: Service Option and Call Number:
- Bit 0: Task Specification:
  - =0 Get the Program Option Word of my task.
  - =1 Get the Program Option Word of the task whose name is contained in R12-13. Option name = OTASK.
- Bits 8-15: Service call number = #53 and the call name = GETPOP.

R12: CAN-code characters 1-3 of the name of the task whose Program Option Word is requested.

R13: CAN-code characters 4-6 of the name of the task whose Program Option Word is requested.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0000	Another task was specified and task is not currently resident or scheduled in the system.
'0001	Another task was specified and your influence limit does not permit you to get the Program Option Word of that task.

REGISTERS CHANGED UPON RETURN

R15: The contents of the task's Program Option Word.

PROGRAMMING CONSIDERATIONS

If the Program Option Word of the caller's task is referenced, there is no need to test the condition code settings upon return.

USAGE EXAMPLES

Get my Program Option Word contents.

```
*        ...
         LDI,R8       GETPOP           SET TASK NAME.
         REX,MAXIV                     REQUEST THE SERVICE.
*        ...                           R15 CONTAINS VALUE NOW.
```

Get the Program Option Word contents of the task named 'MONAMI'.

```
*        ...
         LDI,R12      @MON             SET TASK NAME.
         LDI,R13      @AMI             ...
         LDI,R8       GETPOP+OTASK     SET OPTION/CALL NUMBER.
         REX,MAXIV                     REQUEST THE SERVICE.
         HNR,ERROR                     R15 CONTAINS VALUE NOW.
```

SET NAMED VARIABLE OF SPECIFIED TASK

Call Name: SETVAR Call Number: #54 (REX,MAXIV)

Option Name: OTASK Option Value: #8000

SERVICE PERFORMED

This service adds the Task Variable Name and Value specified to the Task Variables Extension of my task or another task. If a Task Variable already exists in the referenced extension with the same name, replace its current value with the specified value.

CALLING PARAMETERS

R0: Service Option and Call Number:
  Bit 0: Task Specification:
    =0 Calling task is referenced.
    =1 Task whose name in R12-R13 is referenced Option name = OTASK.
  Bits 8-15: Service call number = #54 and call name = SETVAR.

R12: CAN-code characters 1-3 of the name of the task referenced if not calling task.

R13: CAN-code characters 4-6 of the name of the task referenced if not calling task.

R14: CAN-code characters 1-3 of the name of the Task Variable.

R15: Task Variable Value (any 16 bit value)

CONDITION CODES UPON RETURN

NZOC Condition

'1000 - Operation completed successfully.

'0000 - Another task referenced and task is not currently resident or scheduled in the system.

'0001 - Another task referenced and your influence limit does not permit you to modify the Variables of the specifed task.

'0010 - Referenced task has no Variables Extension to its TCB.

'0011 - Variable and value need to be added to Variables Extension but no room exists in the extension to permit their addition at the current time.

PROGRAMMING CONSIDERATIONS

The Variables Extension of a Task Control Block (TCB) is optional. Hence, all tasks do not necessarily have such an extension.

If a task has a Variable Extension to its TCB, the variable named " " with value $8000 is created when the extension is first defined.

This service will set specified variables only. Use the GETASK service to clear the unspecified variable (the unspecified variable is the first variable).

USAGE EXAMPLES

Set the Task Variable named "XYZ" in my task's TCB's Variable Extension to the value -1.

```
*       ...
        LDI,R14         @XYZ                SET VARIABLE NAME.
        LDI,R15         -1                  SET VARIABLE VALUE.
        LDI,R8          SETVAR              SET OPTION/CALL NUMBER.
        REX,MAXIV                           REQUEST THE SERVICE.
        HNR,NOTSET                          XFER IF PROBLEMS.
*       ...                                 VARIABLE HAS BEEN SET.
```

Set the Task Variable named " " in the Variables Extension of the task named "MONAMI" to the value $ABCD.

```
*       ...
        LDI,R12         @MON                SET TASK NAME.
        LDI,13          @AMI                ...
        ZRR,14                              SET VARIABLE NAME.
        LDI,R15         $ABCD               SET VARIABLE VALUE.
        LDI,R8          SETVAR+OTASK        SET OPTION/CALL NUMBER.
        REX,MAXIV                           REQUEST THE SERVICE.
        HNR,NOTSET                          XFER IF PROBLEMS.
*       ...                                 VARIABLE HAS BEEN SET.
```

GET VARIABLE OF SPECIFIED TASK

Call Name: GETVAR Call Number: #55 (REX,MAXIV)

Option Name: OTASK Option Value: #0000

SERVICE PERFORMED

This service returns the value of the specified Task Variable from the Task Variables Extension of the requesting task or another task.

CALLING PARAMETERS

R0: Service Option and Call Number:
Bit 0: Task Specification.
=0 Task referenced is calling task.
=1 Task referenced is task whose name is contained in R12-13.
Option name = OTASK.
Bits 8-15. Service call number = #55 and call name = GETVAR.

R12: CAN-code characters 1-3 of the name of the task referenced if not calling task.

R13: CAN-code characters 4-6 of the name of the task referenced if not calling task.

R14: CAN-code characters 1-3 of the name of the Task Variable referenced.

CONDITION CODES UPON RETURN

NZOC Condition

'1000 - Operation completed successfully.

'0000 Another task referenced and task is not currently scheduled in the system.

'0001 - Another task referenced and your influence limit does not permit you to get variable values from the referenced task.

'0010 - Referenced task has no Variables Extension to its TCB.

'0011 - No variable with indicated name found in Variables Extension.

REGISTERS CHANGED UPON RETURN

R15: The value of the specified variable.

PROGRAMMING CONSIDERATIONS

This service will deliver specified variables only. Use the GETASK service to get the unspecified variable (the unspecified variable is the first variable).

USAGE EXAMPLE

Get me the current value of my Task Variable named "WHY".

```
*       ...
        LDI,R14          @WHY        SET VARIABLE NAME.
        LDI,R8           GETVAR      SET OPTION/CALL NUMBER.
        REX,MAXIV                    REQUEST THE SERVICE.
        HNR,NOTGOT                   XFER IF PROBLEMS.
*       ...                          VALUE IS IN R15.
```

DELETE VARIABLE(S) OF SPECIFIED TASK

Call Name: DELVAR   Call Number: $56 (REX,MAXIV)

Option Name: OTASK   Option Value: $8000

SERVICE PERFORMED

This service deletes the specified Task Variable or all Task Variables from the Task Variables Extension of the TCB of the calling task or another task.

CALLING PARAMETERS

R8: Service Option and Call Number:
- Bit 0: Task Reference Specification:
  - =0 Task referenced is my task.
  - =1 Name of referenced task is in R12-R13. Option name = OTASK.
- Bits 8-15: Service call number = $56 and call name = DELVAR.

R12: CAN-code characters 1-3 of the name of the referenced task if not calling task.

R13: CAN-code characters 4-6 of the name of the referenced task if not calling task.

R14: Deletion Specification:
- o CAN-code characters 1-3 of the named of the Task Variable to be deleted if a single variable is to be deleted
- o The two's complement representation of the value -1 (=$FFFF) if all variables are to be deleted.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Operation completed successfully.
'0000	Another task referenced and task is not currently scheduled or resident in the system.
'0001	Another task referenced and your limit priority does not permit you to delete variables from other task's extension.

'0010 - Referenced task has no Variables Extension to its TCB.

'0011 - No variable of specified name found in extension. Can be set only if a single variable is to be deleted.

PROGRAMMING CONSIDERATIONS

This service cannot delete the unspecified variable. See the GETASK service for getting/clearing the unspecified variable.

USAGE EXAMPLES

Delete all Task Variables from my TCB's extension.

```
*       ...
        GMR,R14,R15                         SET CODE FOR 'ALL'.
        LDI,P8          DELVAR              SET CALL NUMBER.
        REX,MAXIV                           REQUEST THE SERVICE.
*       ...                                 OPERATION COMPLETE.
```

MODIFY ACCESS RIGHTS IN CALLING TASK

Call Name:	MARS	Call Number:	$57 (REX,MAXIV)
Option Name:	OSPACE	Option Value:	$100
Field Names:	ARREAD	Field Value:	$4000
	AREXECUTE		$8000
	ARWRITE		$C000

SERVICE PERFORMED

This service modifies a task's degree of accessibility to a specified region of its own virtual memory.

CALLING PARAMETERS

R0: Service Options and Call Number:

- Bits 0-1: Access Rights Request Code:
  - ='11 Read/Write access. Option name = ARWITE.
  - ='10 Read/Execute access. Option name = AREXECUTE.
  - ='01 Read-Only access. Option name = ARREAD.
- Bit 6: Active Map Load Inhibit:
  - =0 Active map affected is loaded
  - =1 Active map affected is not loaded
- Bit 7: Virtual Region Map Reference:
  - =0 Virtual region specified exists in the task's Instruction Map.
  - =1 Virtual region specified exists in the task's Operand Map. Option name = OSPACE.
- Bits 8-15: Service call number = $57 and call name = MARS.

R15: Virtual Region Description:

- Bits 0-7: Number of the virtual page (0-255) beginning the region.
- Bits 8-15: Number of pages in the region (1-256) specified as the low-order 8 bits of the two's complement, negative representation of the positive number of pages in the region.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	Normal return. Region accessible as requested.
'0000	The region specified does not lie wholly within the virtual memory defined by the referenced map.
'0001	You have attempted to gain a degree of access to a shared region which you are not permitted to have.
'0010	You have specified an Access Rights Request Code of '00 which is not permitted.
'0011	One or more virtual pages in the specified region has no actual page associated with it.

PROGRAMMING CONSIDERATIONS

When this service is invoked, the accessibility of the specified region will not be affected if any condition code but '1000 has been set upon return.

Any degree of access may be gained for private pages belonging to the invoking task.

The degree of access to pages which lie in shared regions may be changed only if the invoking task has established its right to such a degree of access when the shared region was created or inserted. (See Shared Region REX Services Descriptions.)

By established convention, the invoking task may specify a region consisting of all of the virtual memory associated with a map by defining the region to start at Page 0 and consist of 256 pages specified as #0000 in the required notation).

The packed parameter which is used to specify starting page and "negative" number of pages is used in order to conform with the formats of the hardware map-image processing instructions. The DFF 8,8 directive of the MAX IV Macro Assembler may be used to specify such an argument.

USAGE EXAMPLES

I want to have Read/Write Access to the region of my Instruction Map virtual memory beginning with Page Number 48 and consisting of 16 pages.

```
*       ...
        LDI,R15       $30F0                  SET REGION DESCRIPTION.
        LDI,R8        MARS.ARWRITE           SET OPTIONS AND CALL
        REX,MAXIV                            REQUEST THE SERVICE.
        HNR,NOMODS                           XFER IF PROBLEMS.
*       ...
```

Change the access rights of the first 20 pages of my operand addressing space to 'read-only'.

```
*       ...
RGN     DFF           8,8                    DEFINE 2-BYTE DATA WORD
RGNLOW  RGN           0,-20                  DEFINE PAGE 0, 20 PAGES
*       ...
        LDI,R8        MARS!OSPACE!ARREAD     MODIFY OPERAND ACCESS.
        LDM,R15       RGNLOW                 PAGE 0 FOR 20 PAGES.
        REX,MAXIV                            CALL MAX IV.
*       ...                                  IGNORE ERRORS.
```

CREATE SHARED REGION OF MEMORY FROM PRIVATE REGION IN CALLING PROGRAM

Call Name:	CREPRIVATE	Call Number:	$58 (REX,MAXIV)
Option Name:	OSPACE	Option Value:	$100
Field Names:	ARREAD AREXECUTE ARWRITE	Field Values:	$4000 $8000 $C000

SERVICE PERFORMED

This service causes a region of private memory already allocated to the calling task to be designated as shared, so that other tasks can use it. Once designated, the definition of the region is entered into the system's Private Shared Region Directory, and the region remains accessible to the calling task as a Private Shared Region. The region must contain only private pages before the service is called to convert the region to shared pages. The region will remain defined in the system as long as at least one task has the region's definition inserted in its virtual space.

CALLING PARAMETERS

R8: Service Option Bits and Call Number:

- Bits 0-1: Access Rights Request Code:
  - ='11 Read/Write Access. Option name = ARWRITE.
  - ='10 Read/Execute Access. Option name = AREXECUTE.
  - ='01 Read Only Access. Option name = ARREAD.
- Bit 2: Total Write Prohibition:
  - =0 Only the Write Lock determines the Read/Write access of a task to the region.
  - =1 No task other than the calling task is permitted Read/Write Access to the region
- Bit 6: Active Map Load Inhibit:
  - =0 Active map affected is loaded
  - =1 Active map affected is not loaded.
- Bit 7: Virtual Region Map Reference:
  - =0 Specified virtual region exists in the task's Instruction Map virtual memory.
  - =1 Specified virtual region exists in the task's Operand Map virtual memory. Option name = OSPACE.
- Bits 8-15: Service call number = $58 and call name = CREPRIVATE.

R10: Read Access Security specification. May be specified as a Read Access security lock or as a Read Access influence limit. If an influence limit is to be specified, a negative sign (- must precede it. Only tasks that have an influence limit compatible with the specified influence limit will be allowed read or read/execute access to the private shared region. If not negative, then a read security lock (in CAN-code) will be accepted for the created region, and only tasks that can specify a matching key will be allowed read or read/execute access to the private shared region. If R10 =0, then an "always open lock" will be specified which will permit any task to gain read access to the private shared region.

R11. Write Access Security specification. May be specified as a Write Access security lock or as a Write Access influence limit. This argument is specified and used exactly as the read access security specification in R10 but controls the ability of another task to gain write access instead.

R12. Private Shared Region Name (Bytes 1 and 2).
R13: Private Shared Region Name (Bytes 3 and 4).
R14: Private Shared Region Name (Bytes 5 and 6).
R15: Virtual Region Description:

Bits 0-7:	Number of the virtual page (0-255) which begins the region.
Bits 8-15:	Number of pages in the region (1-256) specified as the low-order eight bits of the two's complement, negative representation of the positive number of pages in the region.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	- Normal return. Region is defined as Private Shared Region and accessible as requested.
'0000	- The region you specified does not lie wholly within the virtual space defined by the referenced task map.
'0001	- The region you specified is not free for use as a Private Shared Region (contains all or a portion of one or more other shared regions or contains unallocated pages).
'0010	- The number of pages required for the region exceeds the maximum number of pages which may be used at any one time for shared region definitions as established during SYSGEN by your installation.
'0011	- The name you supplied duplicates that of an existing Private Shared Region recorded in the system directory of such regions.
'0100	- Your task's TCB has no Page Sharing Extension or the extension does not currently have a vacant entry which can be used to record your task's usage of the region.
'0101	- The system as SYSGEN'ed does not contain any provision for a Private Shared Region Directory.

'Ø11Ø - You specified an Access Right Request Code of 'ØØ. Such access right specification is illegal.
'Ø111 - The name specified duplicates that of another Private Shared region currently being accessed by your task. You can already access the region.

PROGRAMMING CONSIDERATIONS

Private Shared Regions - once created - exist for usage as long as at least one task in the system has not terminated its access of the region. When the last task using the region terminates its access, the definition of the region is removed from the system directory.

Unless the creating task intends merely to define the region for usage by other tasks, there is questionable value in requesting other than Read/Write access initially.

Regardless of restrictions imposed on other tasks accessing the region, the creating task is permitted unlimited access to the region. However, if Total Write Prohibition is opted and the creating task exits from the system, no other task will ever be able to have Read/Write access to the region.

The "Active Map Load Inhibit" option is supplied to permit MAX IV Operating System elements to use this service. The option is not effective when specified by an unprivileged task.

If the task which creates a region or uses the region has activated another task to which it plans to pass information using a private shared region, the task should relinquish or suspend itself before exiting until it is sure that the invoked task has inserted the region. Otherwise, the region might be removed from the system directory because the "passer" has exited before the "passee" has inserted the region. Some method of "hand-shaking" is suggested (for example, passer tests bit in private shared region that it has reset -- until passee has set the bit).

USAGE EXAMPLE

Create the Private Shared Region named "OUROP$" for me. The region is to consist of 15 pages and -- in my task -- occupies a region starting at Virtual Page Number 48 of the memory defined by my task's Operand Map. I want to have Read/Write access to the region. If any other tasks want any kind of access to the region, they can have it.

```
*       ...
        ZRR,R1Ø                        SPECIFY NO READ OR
        ZRR,R11                        WRITE LOCKS.
        LFM,R12     PSRNRD             SPECIFY NAME AND OPTION.
        LDI,R8      CREPRIV!OSPACE!ARWRITE
*                                      SET OPTIONS AND CALL NUMBER
        REX,MAXIV                      REQUEST THE SERVICE.
        HNR,NOTDEF                     XFER IF PROBLEMS OCCURRED.
*       ..
PSRNRD DFC         "OUROP$",#3ØF1   REGION NAME AND DESCRIPTION.
```

INSERT GLOBAL SHARED REGION OF MEMORY

Call Name:	INSGLO	Call Number:	#59 (REX,MAXIV)
Option Name:	OSPACE	Option Value:	#100
Augment Names:	ARREAD AREXECUTE ARWRITE	Augment Values:	#4000 #8000 #C000

SERVICE PERFORMED

This service makes a permanently defined and contiguous region of actual memory accessible to the calling task, beginning at the virtual address it has specified. Such regions of memory are defined by name at system generation time.

CALLING PARAMETERS

```
R0:  Option Bits and Service Call Number:
     Bits 0-1:    Access Rights Request Code:
                  ='11  Read/Write Access.  Option name =
                        ARWRITE.
                  ='10  Read/Execute Access.  Option name =
                        AREXECUTE.
                  ='01  Read-Only Access.  Option name =
                        ARREAD.
                        If the region is extended memory,
                        the access rights default to Read/
                        Write access.  An extended memory
                        region is only accessible on a
                        CLASSIC 786x or 787x or CLII/75
    Bit 4:        Local Error Recovery (privileged tasks
                  only):
                  =0   Privileged tasks abort if supplied
                       key does not match the region lock
                       for the degree of access requested.
                  =1   Privileged tasks handle error
                       recovery if a supplied key does not
                       match the specified region lock for
                       the degree of access requested.
    Bit 6:        Active Map Load:
                  =0   Active map affected is loaded
                  =1   Active map affected is not loaded.
    Bit 7:        Virtual Address Map Reference:
                  =0   Specified virtual address exists
                       in the task's Instruction Map.
                  =1   Specified virtual address exists
                       in the task's Operand Map.
                       Option name = OSPACE.
```

Bits 8-15: Service call number = $59 and call name = INSGLO.

R10. Read and Read/Execute) Access key or =0 if task assumes there is no lock.

R11 Write Access key or =0 if task assumes there is no lock.
R12: Global Shared Region Name (Bytes 1 and 2).
R13. Global Shared Region Name (Bytes 3 and 4).
R14: Global Shared Region Name (Bytes 5 and 6).

R15: Virtual Address at which region is to be made accessible to your task. (Low-order 8 bits must be "0".) If extended memory, the value is a "don't care".

CONDITION CODES UPON RETURN

NZOC Condition

'1000 - Normal return. Area is accessible as requested.
'0000 - Region is too large to be defined in specified map at specified address, but region was inserted up to that point.
'0001 - The key you supplied for the degree of access requested does not match the lock defined for that degree of access to the region. (You are a privileged task and specified the the Local Error Recovery option bit B4 of R8 = 1.)
'0110 - Same as 0001 but Access Rights Request Code of "00" was specified.
'0010 - The region of virtual memory of your task in which you wish the definition inserted is not a free region (contains private pages or other shared region definitions).
'0011 - Named region is not defined in system directory.
'0100 - Your task's Page Sharing Extension does not exist or has no vacant entries which can be used to record your task's usage of the shared region.
'0101 - Named region is already accessible to your task (An entry in your task's Page Sharing Extension indicates the fact.)
0110 - Your task is privileged and has specified Local Error Recovery. The access rights specified in R8 are not compatible with the region to be inserted.
'0111 - Global Shared Region is in Shared Multiport Memory (SMM) and SMM is offline.

PROGRAMMING CONSIDERATIONS

If the global region is not an extended memory region, your task's TCB must have a Page Sharing Extension when this service is invoked. This extension must be sized by the TASK/OVERLAY CATALOGER appropriate to intended usage of shared regions.

The virtual region of your task's memory at which you wish to have the shared region made accessible must be available for use.

A single Global Shared Region is not permitted to occupy two different virtual regions simultaneously in a single task.

The "Active Map Load Inhibit" option is supplied to permit MAX IV Operating System elements to use this service to process a task's map image before it is loaded into a hardware map. The option is not effective when specified by an unprivileged task.

USAGE EXAMPLE

I want to access the Global Shared Region named "COMMON" with Read/Write usage permitted. I want the region accessible to my task starting at Virtual Address 16384 (=#4000) in the virtual memory defined by my task's Operand Map.

```
*       ...
        SURD,R10,R10                        SUPPLY DUMMY KEYS.
        LFM,12          GSRNAA              SET REGION NAME AND ADDRESS.
        LDI,R8          INSGLO!OSPACE!ARWRITE
*                                           REQUEST OPTIONS AND CALL NUMBER.
        REX,MAXIV                           REQUEST THE SERVICE.
        HNR,NOTINS                          XFER IF REGION NOT INSERTED.
*       ...
GSRNAA  DFC             "COMMON",#4000
*                                           AREA NAME/VIRTUAL ADDRESS.
```

INSERT PRIVATE SHARED REGION OF MEMORY

Call Name:	INSPRI	Call Number:	#5A (REX,MAXIV)
Option Name:	OSPACE	Option Value:	#100
Augment Names:	ARREAD AREXECUTE ARWRITE	Augment Values:	#4000 #8000 #C000

SERVICE PERFORMED

This service makes a transiently defined region of actual memory accessible to your task beginning at the virtual address you have specified. The region must have been previously "created" by another task or the calling task by designating private pages of that task as "shared".

CALLING PARAMETERS

R0: Option Bits and Service Call Number:

- Bits 0-1. Access Rights Request Code:
  - ='11 Read/Write Access. Option name = ARWRITE.
  - ='10 Read/Execute Access. Option name = AREXECUTE.
  - ='01 Read-Only Access. Option name = ARREAD.
- Bit 4: Local Error Recovery (privileged tasks only):
  - =0 Privileged tasks abort if supplied key does not match the region lock for the degree of access requested.
  - =1 Privileged tasks handle error recovery if a supplied key does not match the specified region lock for the degree of access requested.
- Bit 6: Active Map Load:
  - =0 Active map affected is loaded
  - =1 Active map affected is not loaded.
- Bit 7: Virtual Address Map Reference:
  - =0 Specified virtual address exists in the task's Instruction Map.
  - =1 Specified virtual address exists in the task's Operand Map. Option name = OSPACE.
- Bits 8-15: Service call number = #5A and call name = INSPRI.

R10: Read (and Read/Execute) Access key or =0 if task assumes there is no lock.

R11: Write Access key or =0 if task assumes there is no lock.
R12: Private Shared Region Name (Bytes 1 and 2).

R13: Private Shared Region Name (Bytes 3 and 4).
R14: Private Shared Region Name (Bytes 5 and 6).

R15: Virtual Address at which region is to be made accessible to your task. (Low-order 8 bits must be "0".)

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	- Normal return. Area is accessible as requested.
'0000	- Region is too large to be defined in specified map at specified address, but region was inserted up to that point.
'0001	- The key you supplied for the degree of access requested does not match the lock defined for that degree of access to the region. (You are a privileged task and specified the the Local Error Recovery option bit B4 of R8 = 1)
'0110	- Same as '0001 but Access Rights Request Code of "00" was specified.
'0010	- The region of virtual memory of your task in which you wish the definition inserted is not a free region (contains private pages or other shared region definitions).
'0011	- Named region is not defined in system directory.
'0100	- Your task's Page Sharing Extension does not exist or has no vacant entries which can be used to record your task's usage of the shared region.
'0101	- Named region is already accessible to your task. (An entry in your task's Page Sharing Extension indicates the fact.) If region is shared between CPU's (32/85), a task can have only one shared region at a time in bypass cache.
'0110	- Your task is privileged and has specified Local Error Recovery. The access rights specified in R8 are not compatible with the region to be inserted.

PROGRAMMING CONSIDERATIONS

The task's TCB must have a page-sharing extension when this service is started.

The virtual region of your task's memory, at which you wish to have the shared region made accessible, must be available for use and currently unallocated.

A single Private Shared Region is not allowed to occupy two different virtual regions simultaneously in a single task

The "Active Map Load Inhibit" option is supplied to permit MAX IV Operating System elements to use this service to process a task's map image before it is loaded into a hardware map. The option is not effective when specified by a user task.

The requested region is not available unless some other task still has its definition inserted also. If the calling task becomes the last task that has the region inserted, to pass the region to another task, the calling task must relinquish until the next task has inserted the region.

USAGE EXAMPLE

I want to access the Private Shared Region named "PRIDAT" with Read/Write usage permitted. I want the region accessible to my task starting at Virtual Address 16384 (=$4000 in the virtual memory defined by my task's Operand Map.

```
*       ...
        SURD,R10,R10                    SUPPLY DUMMY KEYS.
        LFM,12        PSRNAA            SET REGION NAME AND ADDRESS.
        LDI,R8        INSPRI!OSPACE!ARWRITE
*                                       REQUEST OPTIONS AND CALL NUMBER
        REX,MAXIV                       REQUEST THE SERVICE.
        HNR,NOTINS                      XFER IF REGION NOT INSERTED.
*       ...
PSRNAA  DFC           "PRIDAT",$4000
*                                       AREA NAME/VIRTUAL ADDRESS.
```

INSERT SHARED LOAD MODULE INTO MEMORY

Call Name:	INSHLO	Call Number:	#5B (REX,MAXIV)
Option Name:	OSPACE	Option Value:	#100
Field Names:	ARREAD AREXECUTE ARWRITE	Field Values:	#4000 #8000 #C000

SERVICE PERFORMED

This service makes a shared region containing a reentrant load module accessible to your task. If the load module has been previously loaded and entered into the system's Shared Load Module Directory, no actual loading operations are performed. If the module has not been loaded and entered into the directory, the module is loaded into task memory. If possible, the definition of the region occupied by the load module is entered into the system directory for use by other tasks. If such a definition cannot be entered, the load module remains accessible to the task in the task's private memory with the degree of access requested.

CALLING PARAMETERS

R0: Service Option Bits and Call Number:

Bits 0-1: Access Rights Request Code:
- ='11 Read/Write Access. Option name = ARWRITE.
- ='10 Read/Execute Access. Option name = AREXECUTE.
- ='01 Read-Only Access. Option name = ARREAD

Bit 2: Total Write Prohibition (ignored if directory entry is not created):
- =0 Only the Write Lock determines the ability of a task to gain Read/Write Access to the region.
- =1 No task, other than the calling task, is to be permitted to have Read/Write Access to the region.

Bit 3: Abort on Load Error Specification:
- =0 If the invoking task does not wish to be aborted if load errors occur in module loading
- =1 The invoking task aborts if load errors occur in module loading.

Bit 4: Local Error Recovery (privileged tasks only):
- =0 Privileged tasks abort if supplied key does not match the region lock for the degree of access requested.

=1 Privileged tasks handle error recovery if a supplied key does not match the specified region lock for the degree of access requested.

Bit 6: Active Map Load Inhibit:
=Ø Active map affected is loaded.
=1 Active map affected is not loaded.

Bit 7: Virtual Region Map Reference:
=Ø Specified virtual region exists in the task's Instruction Map virtual memory.
=1 Specified virtual region exists in the task's Operand Map virtual memory. Option name = OSPACE

Bits 8-15: Service call number = #5B and call name = INSHLO.

R1Ø: Read (and Read/Execute) Access Security specification or lock:

If a directory entry is created, it may be specified as a Read Access security lock or as a Read Access influence limit. If an influence limit is to be specified, a negative sign (-) must precede it. Only tasks that have an influence limit compatible with the specified influence limit will be allowed read or read/execute access to the shared load module. If not negative, then a read security lock (in CAN-code) will be accepted for the region, and only tasks which can specify a matching key will be allowed read or read/execute access to the shared load module. If R1Ø=Ø, then an "always open lock" will be specified. This permits any task to gain read access to the shared load module.

R11: Write Access Security specification or lock:

If a directory entry is created, it may be specified as a Write Access security lock or as a Write Access influence limit. This argument is specified and used exactly as the read access security specification in R1Ø but controls the ability of another task to gain write access instead.

R12: Logical File Name (LFN) currently associated with the real file containing the load module and specified as one of the following forms:

- a 3-character LFN in CAN-code representation.
- "Ø" - which is an alternate specification of the LFN from which the root task was loaded. A resident task uses the LFN specified at system generation by the TASK statement.
- "-1" - which specifies that the last LFN used by the task for task or overlay loading is to be used.

R13: Load Module Name (CAN-code characters 1 to 3).

R14: Load Module Name (CAN-code characters 4 to 6).

R15: Virtual Address at which the load module is to be made accessible to your task. Low-order 8 bits must be "Ø".

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	Normal return. Module inserted as a shared region with the degree of access requested. No load errors occured when the module was loaded.
'1ØØ1 -	Module inserted as a shared region with the degree of access requested. Load errors occurred when the module was loaded.
'1Ø1Ø -	Module inserted as shared region with the degree of access requested after module was loaded and the system directory and page sharing extension entrys were created. No load errors occurred.
'1Ø11 -	Same as '1Ø1Ø except load errors occurred.
'111Ø -	Module loaded and accessible as requested but not as shared region. A system directory and page sharing extension entry would have been created but the region containing the module contained other shared regions or portions thereof. No load errors occurred.
'11ØØ -	Same as '111Ø except the service could not or was not required to create a system directory and page sharing extension entrys.
'1111 -	Same as '111Ø except load errors occurred.
'11Ø1 -	Same as '111Ø but the service could not or was not required to create a system directory and page sharing extension entry and load errors occurred
'ØØØØ -	Region required by the load module is too large to be defined in the referenced map beginning at the specified address.
'Ø1ØØ -	You specified an Access Right Request Code of "ØØ". Such access right specification is illegal. (You are a privileged task and specified the Local Error Recovery Option Bit 4 in R8 = 1.)
'ØØØ1 -	The key supplied for the degree of access requested does not match the lock defined in the directory entry for that region for the degree of access requested. (You are a privileged task and specified the Local Error Recovery Option Bit 4 in R8 = 1.)
'ØØ1Ø -	Load Module named was not found in file specified, or NLSEQUENTIAL specified in the SYSGEN.

REGISTERS CHANGED UPON RETURN

R14. Entry Point Definition Flag
=0 - if no entry point is defined for the module.
=1 - if an entry point is defined for the module.
R15: Module Entry Point
=0 - if no entry point is defined for the module.
= The entry point relative to your task load point if an entry point is defined for the module.

PROGRAMMING CONSIDERATIONS

Shared Load Modules - once inserted - exist for as long as at least one task in the system has not terminated its access of the region. When the last using task terminates its access, the definition of the region is removed from the system directory.

Since a task may insert a Shared Load Module into any region of its virtual space, the contents of a Shared Load Module should be absolute or self-relocating. Only if it is known that all user tasks will insert the module at the same virtual address may such a module be made absolute.

Regardless of restrictions imposed on other tasks accessing the region, the creating task is permitted unlimited access to the region. Since reentrant modules are not to be altered, no task normally requests write access to the region.

The "Active Map Load Inhibit" option is supplied to permit MAX IV Operating System elements to use this service. The option is not effective when specified by an unprivileged task.

If your task's TCB has no page-sharing extension or your installation has not provided resources for the required directory or usage of such modules on a shared basis, the module nevertheless will be usable to your task if not otherwise prohibited by your task's existing environment. The effect of not providing for sharing of otherwise sharable modules will be a loss of actual memory utilization efficiency.

The virtual region of your task's memory at which you wish to have the shared region made accessible to your task must be available for use and currently unallocated; otherwise load errors may occur.

USAGE EXAMPLE

Load the Shared Load Module named "PORTAL" contained in the real file which I refer to as "SM", into the virtual region of my task's Instruction Map memory starting at Virtual Page Number 48. I want Read/Execute access to the module's region. If a directory entry does not currently exist, create one with no Read Lock but preclude any task from ever having the ability to modify the module contents. If loading errors occur, abort my task.

```
*       ...
        SURD,R10,R10                SPECIFY NO READ OR WRITE LOCKS.
        LFM,R12        SLMFNA       SPECIFY LFN, NAME, REGION.
        LDI,R8         INSMLD!#3000.AREXECUTE
*                                   SET OPTIONS AND CALL NUMBER.
        REX,MAXIV                   REQUEST THE SERVICE.
*           ...
SLMFNA      DFC @SM,@POR,@TAL,#3000  LOAD LFN, MODULE NAME, REGION.
```

EXTRACT SHARED REGIONS OF MEMORY

Call Name:	EXTSHA	Call Number:	$5C (REX,MAXIV)
Option Name:	OSPACE	Option Value:	$100
	REGDEF		$800
	XGLO		$8000
	XPRI		$4000
	XSHL		$2000

SERVICE PERFORMED

This service makes a named shared region/load module, or all (any shared regions/load modules occupying all or a portion of a specified region of task virtual memory, inaccessible to your task and permits the associated virtual memory to be reused for other purposes.

CALLING PARAMETERS

R8: Service Options and Call Number:

Bits 0-3: Extraction Scope Specification:
- = '1000 Virtual Global Shared Region(s) is (are) referenced for extraction. Option name = XGLO.
- = '0100 Private Shared Region(s) is (are) referenced for extraction. Option name = XPRI.
- = '0010 Shared Load Module(s) is (are) referenced for extraction. Option name = XSHL
- = '0001 Extended memory Global Shared Region is referenced for extraction.
- = '1111 All (any) shared region/load modules are referenced for extraction.

Bit 4. Extraction Specification Type and.
- =0 Region/load module name is specified in R13-R15
- =1 Virtual Region Description is specified in R15. Option name = REGDEF.

Bit 5: Ignored.

Bit 6: Active Map Load Inhibit:
- =0 Active map affected is loaded.
- =1 Active map affected is not loaded.

Bit 7: Virtual Region Map Reference (ignored if Bit 4 set to zero:
- =0 Specified virtual region exists in the task's Instruction Map
- =1 if specified virtual region exists in the task's Operand Map. Option name = OSPACE.

Bits 8-15: Service call number = $5C and call name = EXTSHA

R13: One of the following:
Global Shared Region Name (Bytes 1 and 2)
Private Shared Region Name (Bytes 1 and 2)
Logical File Name (LFN) of shared load module file
Ignored.

R14. One of the following:
Global Shared Region Name (Bytes 3 and 4)
Private Shared Region Name (Bytes 3 and 4)
Load Module Name (CAN-code characters 1 to 3)
Ignored.

R15: One of the following:
Global Shared Region Name (Bytes 5 and 6)
Private Shared Region Name (Bytes 5 and 6)
Load Module Name (CAN-code characters 4 to 6)
Virtual Region Description:
Bits 0-7: Number of the virtual page (0-255) that begins the region.
Bits 8-15: Number of pages in the region (1-256) specified as the low-order eight bits of the two's complement, negative representation of the positive number of pages in the region.

CONDITION CODES UPON RETURN

NZOC	Condition
'1000	- Normal return. Virtual space is reusable
'0000	- You specified a virtual region which does not lie wholly within the virtual space defined by the referenced map.
'0001	- You are not sharing the region/load module you named.

PROGRAMMING CONSIDERATIONS

By established system convention, a virtual region described as starting at Virtual Page Number 0 and consisting of 256 pages describes the entire virtual space defined by the referenced map.

If Private Shared Regions or Shared Load Modules are extracted by this service and your task is the only task in the system which is accessing the region at the time the service is invoked, the definition(s of the extracted region(s) will be removed from the applicable system directory.

The Active Map Load Inhibit option is provided to permit elements of the MAX IV Operating System to use this service. This option is not normally effective when specified by a user defined task.

USAGE EXAMPLE

Extract all (any) shared regions/load modules which lie totally or in part in the virtual region of my task's Operand Map virtual space which starts at Virtual Page Number #8 and consists of 16 pages.

```
*       ...
        LDI,R5         #30F0                     SET REGION DESCRIPTION.
        LDI,R8         EXTSHA!XCLO!XPRI!XSHL!REGDEF!OSPACE
*                                                SET OPTIONS AND CALL NUMBER.
        REX,MAXIV                                REQUEST THE SERVICE.
*       ...
```

CHAPTER 3
MAX III COMPATIBLE REX SERVICES

3.1 EXCEPTIONS

Most executive services available under the MAX III Operating System are available for execution under MAX IV systems in addition to the unique MAX IV ONLY services described in Chapter 2. The following MAX III services are not available in MAX IV.

SERVICE	NUMBER	PURPOSE
LINKLOAD	#2C	Link load object module
CALL	#2E	Transfer to/from unprivileged state
PUSH	#30	Push registers into privileged stack
PULL	#31	Pull registers from privileged stack
MATCH	#35	Compare byte strings

Most of these services were used only by special parts of the MAX III Operating System and were rarely used directly by user-coded programs. The LINKLOAD and CALL services are provided in MAX IV for batch-processing tasks using a series of overlay loaders invoked by Job Control. The PUSH and PULL services in MAX III were for use by reentrant REX services to ensure their own reentrancy, and the hardware instructions on the CLASSIC Series have replaced their usage. The MATCH service was not used by any MODCOMP standard software, and was rarely used by any customer.

3.2 LIST OF MAX III COMPATIBLE SERVICES

Refer to "Assembly Language Calls for MAX III Executive Services," a chapter of the MAX III GENERAL OPERATING SYSTEM Reference Manual.

DEBUG SERVICE (RESERVED FOR DEBUG PROCESSOR)

Call Name: (none) Call Number: (REX,#1F)

Option Name: (none) Option Value: #80

SERVICE PERFORMED

This service is used exclusively by the DEBUG-processor and is available in the MAX III call format only.

With the option bit reset, this service merely traps to a predefined address, in DEBUG, and stores the address of the REX,#1F call (+1) in a predefined word. This service is used in conjunction with the USERTRAP (#2F) service. The determination of where to trap to and where to store the return address is made based upon whether or not the MAX IV USERTRAP+USERDEBUG service has been called and is currently active.

If the MAX IV USERTRAP call is active, the address to which the system will trap is defined in the DEBUG Trap Table Word -1; and the return address will be stored in Word Ø of that table.

If the MAX IV USERTRAP call is not active, the address to which the system will trap is assumed to be contained in absolute location 1 of the program's operand space, and the return address will be stored at location Ø of that space.

With the option bit set, this service performs the equivalent of the instruction SELECT CURRENT CONDITION CODES (SCCC) except that no registers are destroyed; one inline argument is required which contains the address of a word which contains the condition codes to be returned.

CALLING SEQUENCES

a) Without option: REX,#1F

b) With option: REX,#1F+#8Ø
DFC (address of condition codes)

CONDITION CODES UPON RETURN

NZOC	Condition
'1ØØØ	Always set when option not specified.
'xxxx	User defined codes set when option is specified.

REGISTERS CHANGED UPON RETURN

None.

CHAPTER 4
REX SERVICES THEORY OF OPERATION

The Unimplemented Instruction Trap (interrupt level 4) is requested whenever a program executes a certain class of CLASSIC Series instructions. It is called a trap instead of an interrupt because it aborts the execution of the instruction that caused it instead of interrupting "after" the instruction has completed. Thus, the trap subroutine needs only to examine the dedicated locations ($28 & $49), saved by the interrupt entry, in order to "look at" the instruction that caused the trap and the machine conditions (maps, register block, condition codes) of the program that executed this instruction. Hardware interrupt levels that operate in MAX IV are not allowed to execute unimplemented instructions. Thus, the trap always interrupts a "task" program at a software level (a subdivided part of the CPU's background time) called a task level.

Some unimplemented instructions are optional machine operations such as floating-point. The trap mechanism only works for these instructions when the option is not installed, allowing the trap routine to transfer control to subroutines that simulate (at significantly greater overhead) these instructions when they are not optioned. These simulation subroutines do not execute at the trap level but are "lowered" to the calling task's software priority level as soon as the trap subroutine can determine which subroutine to enter and process a "return link" which will permit the simulation routine to eventually return control to the invoking program.

4.1 REX SERVICES AND THE TRAP

A particular instruction (REX) is never "implemented", and always traps when executed. The MAX IV Operating System uses this mechanism for providing "executive services" to any or all task programs. The trap routine will cause REX to be treated as a special type of "Branch-and-Link" instruction which transfers control to one of several "system subroutines". These eventually return control to the program that executed the REX instruction. These system subroutines always have their entry points in MAP 0 addressing space (Z-space), and most are installed permanently in the operating system's nucleus at system generation time.

The "branch and link" analogy described for the REX instruction can only be carried so far. Actually, REX is much more, since it will allow any task program executing in any virtual addressing space to enter a subroutine in Z-space and eventually return to the original virtual addressing space (I-space) of the calling task. This controlled transfer across map boundaries also lets an unprivileged task enter the privileged state and then return to the state in which it originally existed after the service is finished. As a result of these "inter-map" transfers, the "link" must be a complete Program Status Doubleword (PSD).

The "call number" of the desired "system subroutine" is traditionally selected by a binary-coded field in the REX instruction word. This field is not decoded by hardware and can be defined in any way the trap routine chooses.

```
0                             7   8                             15
---+---+---+---|---+---+---+---  ---+---+---+---|---+---+---+---|
        OP-CODE ($23)            | * |    Subroutine number      |
---------------------------------|---|---------------------------|
                                   |
                               Option Bit
```

Figure 4-1. MAX III REX Instruction Format

4.2 TWO TYPES OF REX SERVICES

In the MAX III Operating System, this 8-bit field is broken down into a 7-bit subroutine number (128 maximum) and a one-bit "option" argument (which can be passed to the selected subroutine number for use as it chooses). A table containing the entry points of these "MAX III" services subroutines is maintained in the operating system's nucleus. These entry points are ordered by service number. The trap subroutine must extract the service number from the REX instruction that caused the trap and use it as an index to this table of entry points (always in the 8-space). Control is immediately transferred to the subroutine's entry point while at the same time the trap level is cleared. Thus the service subroutine executes at the priority level of the program that called the REX instruction.

In MAX IV systems a similar scheme is followed, but two types of calls for each REX service are supported. A MAX IV service call and a MAX III compatible service call. A MAX III service in MAX IV uses an entry scheme similar to that of the MAX III Operating Systems. Most, but not all, MAX III REX services and options are supported, and the actual service entry is as in MAX IV. The calling registers are stored in the stack and R1 is destroyed in the subsequent service entry. The MAX IV scheme uses a particular REX service number ($32) to create up to 256 additional REX services. A larger table supports the entry points of these MAX IV services. A system can be configured with only the MAX IV type services, or it can be configured with both types of services to allow tasks written under MAX III systems to execute under a MAX IV system. Currently, all system batch processors compilers, assemblers, etcetera use MAX III services. Thus, such services are still generally required in MAX IV systems. Only 128 additional MAX IV types of REX service are currently implementable.

Any REX service may be called by any task at any time. This requires that it be reentrant. It must modify only memory cells that are unique to the calling program. Two push stacks in each task are maintained for use by REX services that are called by that task, thus providing these services with the scratch memory cells necessary to permit them to be re-entrantly coded and used recursively

The major philosophic differences between MAX IV type services and MAX III compatible services are the way in which they are called, and the way in which they use general registers. Both types of service subroutines use the register block (RB) being used by the calling task so that arguments may be passed to/from such services using the registers. Thus, this register block serves as sort of a "global common" between the two shared addressing spaces (I-space and O-space). MAX IV type services attempt to use only register arguments, when possible, for the sake of efficiency. MAX III compatible services follow the conventions used by the older operating systems (for example, many calling arguments are located "inline" in memory cells of the I-space following the REX instruction and must be skipped over upon return to the calling program).

MAX IV type REX services do not reference the REX service number and option bit fields of the REX instruction, using instead a service number and option field passed in a dedicated general register (R8). Refer to Figure 4-2. The table below and a flow-chart presented later will summarize the differences and interrelationships of the two types of services executable under MAX IV.

```
0                                 7 8                              15
+-----------------------------------------------------------------+
|        OP-CODE ($23)              |             $32              |
+-----------------------------------------------------------------+
```

Figure 4-2. MAX IV REX Instruction Format

QUESTIONS	ANSWERS: MAX III "COMPATIBLE" SERVICES	ANSWERS: MAX IV "EFFICIENT" SERVICES
Uses REX instruction's service number field as index to entry point?	YES	NO, uses argument in R0 (lower byte)
Uses REX instruction's option bit field?	YES	NO, uses arguments in R0 (upper byte) and has up to 8 unitary/binary option bits/fields.
Uses inline memory arguments and argument addresses?	YES	NO, registers only, or memory arguments pointed to by address pointers in registers.
How are alternate returns indicated?	Inline branch address arguments	Condition codes testable upon exit from service.
Saves/restores registers modified within service subroutines?	YES	YES

Table 4-1. Summary of Service Type Differences

4.3 THE TRAP SUBROUTINE

A description of how the MAX IV Unimplemented Instruction Trap (UTRP subroutine is coded follows. The processing of trapped REX instructions and unimplemented "macro" instructions (when simulated are discussed. Trapped macro instructions are processed in a manner similar to the REX instruction except that the operation code of "macro" instructions (bits 0-7 is used to select the simulation subroutine and the remaining fields of the instruction word (usually two 4-bit register addresses must be passed as arguments to the simulation routine. Refer to the Priority vs Time in Figure 4-3 and the flowchart in Appendix A during the following discussion. The "letter" keys in parentheses are keyed to Figure 4-3.

To follow the action of the Unimplemented Instruction Trap routine, assume that some unprivileged task, executing at task priority level 128, and located in some virtual addressing space (I-space) other than the one defined for the system (Z-space), has just executed a REX instruction at point (A). A hardware trap is generated automatically to hardware interrupt level #4, and a hardware context switch occurs. This immediately selects some general register block, other than the one being used by the interrupted program. (The trap routine uses several registers in dedicated general register block number #F, which it shares with the Taskmaster, the memory parity trap, and the IDLE task.) The trap routine is entered (B). Since the trap routine has its own dedicated register block, it need not save the registers of the interrupted program.

Since programs executing at hardware interrupt levels are not permitted to execute unimplemented instructions, the trap routine first checks to see if this has been attempted. The system will not process such a software failure in any program. HLT is used to prevent further degradation of the system, which would hide the reason for the failure.

Highest Priority

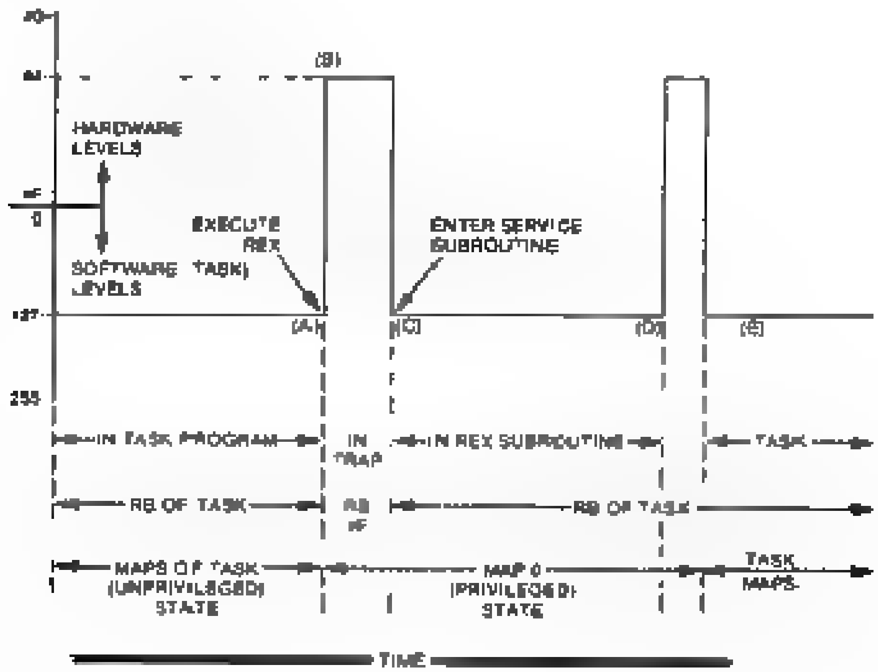

Figure 4-3. Priority -vs- Time During REX Service Invocation

4.3.1 REX LINK

The next job of the Unimplemented Instruction Trap routine (4.UTR) is to build the "REX LINK". This link is the Program Status Doubleword (PSD) of the calling task program at the instant before it was trapped executing the REX instruction. It will be used eventually to return control to that task, and in the interim, it will be used to find the REX instruction that caused the trap. The PSD consists of two 16-bit values: The "old" contents of the PR register (the virtual address of the trapped instruction), and the "old" contents of the PS register (the status fields indicating which maps, register block, and condition codes were in use by the trapped program).

Because it is not desirable to destroy any of the general registers of the calling program, we must dispose of this link into the special REX LINK PSD stack provided for each task. Such action also allows the REX services to automatically nest to various levels in a LIFO manner.

This REX LINK PSD of the call is obtained from the trap's dedicated interrupt vectors (cells #28 and #49 in Z-space). The hardware trap mechanism saves the return PSD in those locations.

The first action performed on the REX LINK PSD is to preset the condition codes in the PS part. Thus, all REX services will exit with these preset condition codes unless the service subroutine takes action to modify them. Although this prevents the condition codes of the calling program from being returned to the calling program, it ensures that these condition codes will be set to a known state as "returned subroutine status arguments" to the calling program. Every other instruction changes condition codes, and so should the execution of a REX service. A default value of NZOC ='1000 indicates a normal return from all MAX IV REX services, and this value is setup initially by the trap routine. Of course, instruction simulations, such as floating-point, will formulate a condition code suitable for the simulated instruction.

The PR part of the link is incremented by one so that it will point to the return address in the calling program's virtual I-space or to the address of the 1st inline argument if such arguments exist (MAX III compatible calls only). It is the responsibility of the REX service subroutine or unimplemented instruction simulation routine to increment this PR link so as to skip any inline arguments or instruction words before control is returned to the caller.

The trap subroutine will further edit another version of the original PS for direct entry to the REX service routine to be selected. This version will retain the register block of the original program, but the map fields will be forced to MAP 0 to select the Z-space where all REX service subroutines reside and the condition codes will be set to NZOC='0000. The privileged state will be forced since all REX subroutines may need to execute privileged instructions. The original PR link will be used to

fetch the trapped instruction from the virtual I-space of the trapped program and process this instruction as follows:

- o If the trapped instruction is a REX,MAXIV (#2332), then the trap subroutine will prepare to select one of the MAX IV type of services from the table at address UCBIV.

- o If the trapped instruction is any other REX operation (#23xx where xx not equal #32), the trap subroutine will prepare to select one of the MAX III compatible type services, if they have been optioned, from the table at address UCBIII.

- o If the operation code is not REX, it is examined further, and if it is one of three optional instruction operation code groups (1X, 3X, 5X) an instruction group simulation routine will be selected for entry. If not a 1x, 3x, or 5x operation code, the user will be aborted.

For MAX III type services, or instruction group simulation routines, the desired entry point is selected while the trap subroutine is still active at level 4. MAX III type services use the low order seven bits of the trapped REX instruction as an index to select the entry point from the table (UCBIII) of all such service entry points. The instruction simulation entry points use bits one and two of the instruction operation code as a binary index to a table of three routines.

MAX IV type services select their entry point after the trap level is cleared and use the task's RB and priority level.

4.3.2 TASK LEVEL

The selected entry point (PR) and the edited status (PS) are now used in conjunction with a special instruction "CIR,R" to clear the trap level, drop to the software task level of the calling program, reselect the general registers of that program, save all 15 general registers on the task's main stack, and enter the selected subroutine in the X-space (key (C in Figure 4-3). General register R1 is lost in the entry process, but if required, it can be recovered from the main stack.

MAX IV type calls are similar. After clearing the trap level with a "CIR" the 15 general registers of the calling task are "pushed" into the main stack which is located in the task's TCB. Then the contents of bits 8-15 of general register 8 are used to select the service entry point from the table (UCBIV) of such entry points. General register R1 is destroyed in this selection process, but the original R1 is in the main stack should it be needed by the service subroutine or need to be returned to the calling program.

The final chosen REX service subroutine may proceed to execute in a privileged manner to perform the desired service. It may use the special REX macros and utility subroutines to obtain any information it needs from the calling program. When it is finished, it will return to the calling program by branching to an appropriate point in I4,XIT (D).

REX services may wish to call themselves or other REX services. This is automatically made possible by the LIFO nature of the REX LINK PSD stack maintained by I4.UTR and I4.XIT.

The PS/PR push stack is smaller than the main push stack and may only be used for saving and restoring REX LINK PSD's. This special push stack permits the system to manage the nesting of levels by knowing how "deep" a particular task is in this stack at any time. The MAX III Operating Systems used REX,PUSH and REX,PULL services to perform this maintenance function at considerably greater overhead than required in a CLASSIC. For this reason, these REX services have been deleted from MAX IV systems, and the hardware PSM and PLM instructions are used instead.

4.4 TYPICAL REX SERVICE SUBROUTINE IMPLEMENTATION

Most REX services are implemented in the form of a "closed-subroutine" in which the entry arguments and return arguments are processed internally in a manner suitable for direct interface to a "MAX IV type" of call.

When a "MAX-III-type" of call also exists for the same service, then an entry interface and an exit interface are implemented which are coupled to the basic closed subroutine implemented for the "MAX IV type" of call.

The MAX III entry interface must fetch the calling arguments in a manner suitable for this particular service. It must then reformat them, as necessary, so that the interface can invoke the "heart" of the MAX IV service. In general, this involves fetching inline memory arguments in the I-space of the calling program.

The MAX III exit interface must accept the return arguments from the MAX IV service subroutine and then reformat them, as necessary, for return to the calling program in a format suitable for that MAX III service call. This involves the testing of condition codes as they are returned by the MAX IV service subroutine and then converting them into appropriate error-processing responses for return to the calling program. Most MAX III type services have an inline argument that is the address of an error processing routine where transfer of control must be made should errors occur.

4.4.1 REX SERVICE NAMES

A naming convention for the entry points of all types of REX services is established by the MAX IV System Generation process. For any service with a unique name "xxxxx..." of at least 4 characters, the following system names must be assigned.

```
M$xxxx  ...for a "resident" MAX IV type service.
M$$xxx  ...for the "resident" MAX III interface of a service.
N$xxxx  ...for a "nonresident" MAX IV type service.
N$$xxx  ...for the "nonresident" MAX III interface of a service
```

Notice that four characters (xxxx of the basic service name must be unique for all MAX IV types, and that only three characters xxx) must be unique for all MAX III type interfaces. The "resident nonresident" designation may be somewhat of a misnomer since all service entry points must terminate at some fixed resident part in the MAX IV nucleus, but, the "nonresident" type of service may invoke the loader to bring in all or part of the service routines from the system nonresident module file (SM). The "resident" type will perform all service routines with strictly resident elements.

4.4 2 RESIDENT/NONRESIDENT

Many standard executive services may be selected as either a "resident" or a "nonresident" version: The former being used when the utmost in speed and reliability is required, the latter being used when better memory utilization is desired, and the service execution time for that particular service can tolerate slightly longer execution time. "Nonresident" services are suitable only if a dependency upon a secondary storage device can be tolerated for all users of the service.

A typical executive service (Binary Hex to ASCII-String conversion service) is shown below. Only the "resident" version is illustrated.

Example 4-1: TYPICAL REX SERVICE

```
       PGM          HX.III       MAX III VERSION OF HEX SERVICE
       INT          M$$HEX
       EXT          CV$HEX
       EXT          I4$XIT       REX SERVICE RETURN
       INS          MC,IVEQUATES REX PARAMETER SYMBOLICS
       INS          MC,IVMAC     INSERT DATA STRUCTURES EQUATES
***** BINARY INTEGER-TO-HEXADECIMAL ASCII CONVERSION SERVICE
* ENTER WITH:
*      (R3)=16 BIT INTEGER
* REX CALL FORMAT IS:
*      REX,#3B
* EXIT WITH:
*      (R1),(R4-R15)-UNMODIFIED
***** (R2,R3)=BYTES 0-3 CONTAIN ASCII HEX DIGIT CHARACTERS 0-3
M$$HEX EQU $       <<-----<<    ENTRY POINT OF SERVICE
       TRR,R13,R3               TRANSFER ARG TO MAX IV WAY
       BLM,R9      CV$HEX       CONVERT TO HEX IN COMMON SUBROUTINE.
       LDM*,R1     LCBSPT       GET REGISTER SAVE STACK ADDRESS
       STMD,R12,R1 1            SAVE RESULT FOR RETURN IN R2,R3
       BRU         I4$XIT       REX SERVICE RETURN
       END

       PGM         HX.IV        MAX IV VERSION OF HEX SERVICE
       EXT         I4$XIT       REX SERVICE RETURN
       INT         M$HEX        MAX IV REX SERVICE ENTRY POINT
       INT         CV$HEX       ENTRY TO CONVERSION SUBROUTINE
       INS         MC,IVMAC     INSERT MAX IV DATA STRUCTURES EQUATES
       INS         MC,IVEQUATES REX PARAMETER SYMBOLICS
```

```
***** BINARY TO HEXADECIMAL ASCII STRING CONVERSION
* ENTER WITH:
*       R13            =16-BIT NUMBER TO BE CONVERTED(UNSIGNED)
* CALL FORMAT IS:
*       R8             =CALL NUMBER (#3B), NO OPTIONS
*       REX,MAXIV
* EXIT WITH:
*       R12            =BYTES 0 & 1 OF STRING
***** R13            =BYTES 2 $ 3 OF STRING
M$HEX EQU $            <<----<<    REGISTERS ALREADY IN STACK, R1 DESTROYED
      BLM,R9           CV$HEX      CONVERT TO HEX IN COMMON SUBROUTINE
      LDM*,R1          LCBSPT      GET REGISTER SAVE STACK ADDRESS
      STMD,R12,R1      11          SAVE RESULT FOR RETURN IN R12,R13
      BRU              I4$XIT      REX SERVICE RETURN
*
*THE "HEART" OF CONVERSION SUBROUTINE--COMMON TO BOTH REX SERVICE FORMS
*
CV$HEX EQU $          <<-----<<    SUBROUTINE BLM,R9) ENTRY POINT
      LBR,R12,B0                   SET FLAG WHICH WILL INDICATE WHEN DONE
XX    TRR,R14,R12                  SET FLAG TO INDICATE WHEN WORD FINISHED
AA    LLS,R12,4                    SHIFT NIBBLE
      XBR,R14,B0                   TOGGLE FLAG SO LOOP PERFORMED TWICE
      LLD,R12,4                    SHIFT BYTE DOUBLE (GET NEXT HEX DIGIT)
      ADI,R12           #36        CONVERT TO ASCII
      TBRB,R12,B9       ZZ         BRANCH IF (A-F)
      SUI,R12           #7         NO, MAKE INTO (0-9)
ZZ    ABR,R12,B15
      TBRB,R14,B0       AA         BRANCH IF BOTH BYTES IN WORD NOT FINISHED
      TBRB,R14,B8       XX         BRANCH IF ALL FOUR BYTES NOT FINISHED
      TRR,R13,R12
      TRR,R12,R14                  SET UP RETURN ARGUMENT IN CORRECT REGS.
      BRX,R9                       RETURN TO APPROPRIATE SERVICE INTERFACE
      END
```

In this example, note that the MAX III call and the MAX IV type call both use a common subroutine, CV$HEX, which is called by each with a "BLM,R9" instruction. The MAX III type interface is separately assembled so that it can optionally drop out of the system if no MAX III services are optioned during system generation.

The MAX IV type routine (M$HEX calls this common subroutine directly since that subroutine is designed to process all entry arguments in general registers in exactly the same format as delivered by the MAX IV type of call. When the subroutine returns control to the MAX IV type routine, the return arguments are also in a form suitable for this type, and are in R12 and R13. All other registers of the original calling program are not yet restored, but must be. This is accomplished by storing only the two return register arguments in the main stack of the task (in the words where R12 and R13 of the calling program were originally saved). All registers are then pulled from the stack save area, and will be restored except the new, surviving arguments in R12 and R13. The register pull and return to the REX invoker are done in routine I4$XIT. I4$XIT is one of several aids provided to facilitate REX implementions and is described in subsequent sections of this chapter.

The MAX III type entry interface (M3$HEX) moves the calling argument from R3 (the MAX III way, to R13 for use by the common subroutine which is then called directly.

The MAX III type exit interface (CMCTA) must now move the return arguments (R12,R13) to their MAX III position (R2 and R3). This is done as in the MAX IV type call. Since this particular service has no error conditions or alternate exits, no further processing is necessary. The service returns to the invoker by using routine I4$XIT, the REX service exit routine.

The common subroutine (CV$HEX) is coded as any closed subroutine would be coded. Some conventions have been established, however, to facilitate consistent and efficient REX service implementation.

4.4.3 ERROR CONDITIONS

Since all MAX IV type services return error indications through the condition codes of the calling program, the common subroutine must keep this in mind. Since such condition codes will not actually be returned until the final REX LINK is restored, the common subroutine must process these condition codes directly in the REX LINK PS. The MAX III type exit interface will likewise examine these condition codes and will then convert them into the appropriate action expected by the calling program. Actually, the final condition codes are returned intact to even a MAX III type service caller, but no user should ever attempt to test them since such action could not be performed in a MODCOMP II or MODCOMP III computer. Thus, usage of MAX III services would not be universal across MAX III/IV if condition codes are tested upon return.

4.5 CONVERTING EXISTING MAX III REX SERVICES

If a user has an investment in computer systems that use only the MAX III Operating System and wishes to upgrade to a MAX IV Operating System, then that user is faced with conversion of these old custom services. Although most task programs written for the older operating systems do not require conversion to run on MAX IV Systems, custom REX services must be converted.

The major difference encountered by a MAX IV REX service (or a MAX III compatible service written to operate on MAX IV) is that it can not directly access arguments in the calling program. This is because in the process of entering a REX subroutine, the hardware map s used to effect the virtual addressing space(s) of the calling task have been switched-off. The entire resident nucleus of the operating system is mapped in the I-space defined by MAP 0 and this space contains all executive service subroutines, interrupt routines, and data structures of MAX IV. The data structures of MAX IV are totally different from those used in MAX III Systems, which is another major reason why conversion is necessary.

4.5.1 PASSING ARGUMENTS

For the reasons above, most standard MAX IV services pass all arguments through general registers. If the MAX III service to be converted for execution on MAX IV has used inline memory arguments, it is faced with using special instructions to look back into the virtual addressing range(s) of the calling program and to get the desired arguments. The techniques for doing this are simple, but require explicit conversion of old services at any point where arguments are passed between service and calling programs. Additional services are provided to assist the REX service with such tasks as switching maps, testing and setting return condition codes and returning to the invoking task.

4.5.2 SAVE/RESTORE REGISTERS

Another difference experienced by a MAX III service in a MAX IV environment is that the REX.PUSH and REX.PULL services are not provided by MAX IV and are not needed. Instead, MAX IV services use the hardware "PUSH/PULL" instructions (PSM and PLM) of the CLASSIC to allocate/deallocate re-entrant temporary cells and save/restore registers. MAX IV provides each task with two push stacks. The larger stack is used just as is the single stack in MAX III, except that REX LINK'S are not saved/restored in this stack. Instead a second smaller stack, called the PS/PR stack is used for pushing REX LINK information. More efficient use of stack space and easier system debugging results from separation of these stacks and use of hardware push/pull instructions. A pointer to the control table of each stack is maintained in dedicated memory cells in the Lower Control Block (in Z-space). These cells are updated each time the Taskmaster switches tasks so that they always reflect the current stacks of the current task.

4.6 IMPLEMENTATION OF CUSTOM REX SERVICES

A group of REX services is supplied as part of the MAX IV Operating System. In addition, the system user may also choose to design, implement, and install REX services to meet specific installation requirements.

Writing one's own REX service is not to be taken lightly, because the service author must realize that such a service is a re-entrant, privileged subroutine in the system nucleus (Z-space), capable of doing anything with any task or with the operating system itself, including its destruction.

However, for the system programmer who must add global re-entrant services, that must do privileged things, a custom REX service is easily written and added to the system at system generation time. Thorough debugging of this service must follow before system integrity can be guaranteed if the services are to be made callable by unprivileged users. The services must be prepared to handle any possible arguments presented to them by the calling task.

REX services may be configured to be permanently resident in the virtual/actual memory of the MAX IV Operating System Nucleus. While such residency provides optimum response time to a service request by an invoker, the memory in which resident services reside is unavailable for other usage during system operation.

Where a large number of REX services are configured in a system and/or the amount of actual memory available is limited, the amount of memory committed for residence of REX services may be so large as to degrade memory usage and/or other performance characteristics of the system.

Consequently, the MAX IV Operating System provides a mechanism which permits user written REX services to be used in nonresident form. In this context, nonresident means that the program constituting the service is not permanently resident in memory but occupies memory only when the service is in active use.

All REX services supplied with the MAX IV Operating System are provided in resident form, because the operation of the system does not permit the services to be nonresident. The user's REX services may be configured as either resident or nonresident at the time the SYSGEN process is performed for the installation. Refer to the MAX IV SYSGEN, System Guide Manual listed in the Preface. (Only resident standard services are supplied with current versions of MAX IV, although the mechanics for implementing custom nonresident services are present and operable.)

Once an installation has chosen to configure at least one REX service in nonresident form, all REX services which have a nonresident form and which have not been installed as resident REX services may be made available to tasks in the system with no further significant memory usage penalty.

The following sections describe.

- o The linkage mechanisms by which control is passed between the invoking task or REX service and a resident REX service.
- o The linkage mechanisms by which control is passed between the invoking task or REX service and a nonresident REX service.
- o The system environment that exists when a nonresident REX service initially receives execution control.
- o Programming considerations that are of significance when designing and implementing nonresident REX services.

Following these definition sections, an example of a custom REX service in resident form is specified and implemented.

6.1 REX SERVICE IDENTIFICATION

A specific REX service is identified by a pair of hexadecimal digits. The MAX IV Operating System supports two different methods of specifying the service to be invoked:

- o The standard MAX IV linkage convention in which the number of the REX service to be invoked is specified by placing the number in Bits 8-15 of Register 8 and invoking the service through a REX instruction having the form #2332

- o The MAX III compatible linkage convention in which the REX service is invoked using a REX instruction of the form "#23xx" where "xx" is replaced by the number of the service. No MAX III compatible service has been assigned the identifying number #32.

6.2 RESIDENT REX SERVICE LINKAGE MECHANISMS

6.2.1 Invocation of a REX Service by a Task

A task invokes a REX service by attempting to execute a REX instruction. The attempt to execute the instruction is trapped and control is passed to the MAX IV Unimplemented Instruction Trap Processor. Upon recognizing the REX instruction operation code, the Processor:

1. Increments the trapped instruction address by 1. The address is called the REX PR Link.

2. Initializes the Condition Code field of the trapped program status word to '1000. The status word is called the REX PS Link.

3. The resultant REX PS and PR values are pushed onto the PS/PR stack pointed to indirectly by LCBPST.

4. The trapped REX instruction is examined and one or the other of the following actions is taken:

   - o If the trapped instruction was #2332, the contents of Registers 1 through 15 are pushed to the task's Main Push Stack. The contents of bits 9-15 of Register 8 replace the corresponding bits in Register 1 and the remaining bits of Register 1 are cleared to 0. The contents of Register 1 are then used as an index to select a REX service entry point from the SYSGEN-constructed table called UCBIV in Z-space.

   - o If the trapped instruction was not #2332, the contents of bits 9-15 of the trapped instruction are used as an index to select a REX service entry point from the SYSGEN-constructed table called UCBIII in Z-space. This entry point is placed into TCBCRX. General registers 1 through 15 are then pushed into

the task's main stack. The contents of R1 are altered in the actual entry to the REX service.

5. An initial program status word is constructed for entry to the selected REX service. The status word indicates:

   - MAP Ø (Z-space) as Instruction Map and Operand Map.
   - Use of the Register Block assigned to the invoker.
   - Privileged state of operation.

6. Control is passed to the REX service program using the entry point address selected and the program status word constructed.

If a specific REX service is selected that is not configured in the installation's system or if an invalid REX number is specified, the procedure passes control to a standard processor which handles these cases. A flow chart of the above described procedure may be found in Appendix A.

4.6.2.2 Invocation of a REX Service by a REX Service

A REX service invokes another REX service in the same fashion as a task.

The maps and register block in the return PSD are those that are being used at the time the REX is executed.

4.6.2.3 Return of Control to the Invoker by a REX Service

The REX service returns control to the invoker by restoring registers, if necessary, and executing a branch to a REX service return routine such as I4$XIT (described below).

4.6.2.4 Linkage Data Structures

During the initial phases of SYSGEN, the user specifies which REX services are to be included in the system. In the SYSGEN process, all REX services are identified by a service name whose first three characters are unique in the MAX IV Operating System.

To include the specified services, the initial phases of SYSGEN generates EXTernal labels for each service to be included. These labels take one of the following forms:

- "M$xxxx" - for a resident MAX IV REX service.
- "M3$xxx" - for a resident MAX III compatible REX service.

  where "xxxx" or "xxx" are replaced by the first four or three characters of the service name.

These EXTernal labels are generated in such a fashion that when the reference is satisfied during the link-edit phase of SYSGEN, the service entry address table(s) referred to above will be generated.

4.6.3 NONRESIDENT REX SERVICES

Nonresident REX services employ the same linkage mechanisms that are used by resident REX services. However, the program constituting a nonresident REX service exists as one or more cataloged load modules.

In order for the service to execute when invoked:

1. Virtual and actual memory in the S-space must be allocated to load the first segment of the REX service program body.

2. The loading process must be performed.

3. The user must pull two words from the PS/PR return stack (the nonresident flags using REXPRP and save them in a re-entrant fashion. This will allow the nonresident REX to use the REX macros.

4. Before branching to the UCB vectors to return to the REX invoker, load a secondary overlay or call another nonresident REX, the nonresident REX must restore the two nonresident flag words saved in (3) by using REXSRP.

When the REX service has completed execution, the virtual space and actual memory that it occupied must be freed for reuse by other system elements and tasks. To permit the use of nonresident REX services, the MAX IV Operating System provides a set of re-entrant subprograms that perform all of the required linkage support functions.

The following sections discuss these standard service subprograms and the manner in which these subprograms are interfaced into the normal REX service linkage mechanisms.

4.6.3.1 Non-Resident REX Service Linkage Programs

There are five distinct entry points to the nonresident REX service linkage programs. These entry point addresses are located in fixed words of the Upper Control Block (UCB), and their functions are as follows:

- UCBSRE Performs memory allocation of one page for loading and entry to standard MAX IV nonresident REX services.

- UCBCRE - Performs memory allocation of one page for loading and entry to MAX III-compatible services.

- UCBSRX - Normally provides memory deallocation and return of control from a standard MAX IV service.

- UCBCRX - Normally provides memory deallocation and return of control from a MAX III-compatible service.

o UCBNRO - Provides for loading overlays of the original REX service module in the page already allocated

The following differences exist between the services provided at the entry points:

1. The routines whose addresses are in UCBSRE and UCBCRE expect to be entered after all 15 registers have been pushed to the task's Push Stack and Register 1 has been altered.

2. The routine whose address is in UCBSRX expects the task's Push Stack to contain the contents of 15 registers to be restored prior to returning control.

3. The routine whose address is in UCBCRX expects the task's registers to have been restored.

4.6.3.2 Non-Resident REX Service Linkage Mechanisms

Direct Service Entry - If, during SYSGEN, you specify that a specific service is to be nonresident or that all services not included as resident are to be included as nonresident , the SYSGEN process will generate EXTernal labels of the form N$xxxx or N$$xxx for such services ,instead of the M$xxxx or M$$xxx forms .

All labels of the form N$xxxx are INTernalized to be EQUated to the entry point whose address is in UCBSRE. Similarly, all labels of the form N$$xxx are INTernalized to be EQUated to the entry point whose address is in UCBCRE.

Hence, the link-edit phase of SYSGEN causes the addresses of the entry points of the appropriate linkage service programs to be placed in the tables used by the Unimplemented Instruction Trap Processor and the required linkage programs to be included.

As a consequence, when a nonresident REX service is invoked by a task or another REX service, the nonresident linkage program required is entered to effect the entry to the REX service.

Indirect Service Entry - It is sometimes useful or necessary to preface the execution of the entry linkage programs with resident segments of code that alter the environment prior to entry of the standard nonresident service linkage routine.

If such action becomes necessary, the preface code may be assigned an INTernalized name of the form N$xxxx or N$$xxx and placed in one of the libraries (for example, file UL, searched during the link-edit phase of SYSGEN.

Because of the fashion in which the link-edit process is performed, the first INTernalized label that is encountered will be used to satisfy EXTernal references and subsequent occurrences of INTernalized labels of the same form will be ignored. Hence, a preface section of code can always be used to replace the equivalent direct entry point INTernalized label.

Preface sections of code may then invoke the appropriate direct entry point through a branch (indirect) instruction such as: BRU* UCBSRE

Service Exit - When a nonresident REX service has completed execution and is prepared to return control to the invoker, the service may always pass control to one or the other of the nonresident exit linkage routines whose addresses are stored in words UCBSRX and UCBCRX in the Upper Control Block.

4 6.4 NONRESIDENT REX SERVICE ENVIRONMENT

A nonresident REX service operates in a system environment very much similar to that of the resident REX service. That is, when a nonresident REX service receives control to begin execution:

1. Its program status is the same as it would be if the service were resident.

2. REX PS and PR links are in the appropriate positions in the PS/PR stack The first 2 words are the nonresident flag. To use the REX macros, those 2 words must be pulled from the PS/PR stack upon REX entry and pushed back onto the PS/PR stack before branching to the UCB vectors to exit or load or secondary overlays, or calling another nonresident REX

3. Register contents upon entry are identical unless modified by prefacing code

However, the virtual/actual memory allocation performed as part of the entry linkage will result in one page (256 words) of memory in an available region of Z-space being provided for loading and residence of the service program. The memory allocated becomes an extension of the TCB of the invoking task and of no other task. As a consequence of this form of memory allocation, a nonresident REX service.

1. May be but does not need to be re-entrantly coded, yet only one copy, of the service need be cataloged for all tasks that might invoke it.

2. Is limited to a maximum of 256 words of simultaneously resident code ,although, with the precautions specified under Programming Considerations for nonresident services in the following section, it may load its own overlays as required.

3 Can manipulate the returned condition codes only by using the appropriate REX macro described below.

4. Does not have the access to the MAP @ REX subroutines that a resident task does.

5. To select an invoker's IM or OM for access:

```
SIA,#P          No Context Switches
REXFPS  reg     FETCH INVOKER'S PS
SOOM,reg        or SIOM,reg  SELECT MAP
RIA,#P          Allow Context Switches
 user code
SZOM            RESELECT EXEC OM
```

4.6.5 PROGRAMMING CONSIDERATIONS FOR NONRESIDENT SERVICES

4.6.5.1 Module Naming

The initial load module of a nonresident REX service must be cataloged under a name in one of the following formats:

- o NR$Sxx for standard MAX IV linkage convention services
- o NR$Cxx for MAX III-compatible linkage convention services

  where xx is replaced by the pair of hexadecimal digits that uniquely identify the service

Overlays used by nonresident REX services must be cataloged under a name in the format NR$xxx where xxx may be replaced by any 3 CAN-coded characters that will not duplicate the name of the initial load module of any nonresident REX service. Installations that choose to design and implement their own nonresident REX services may avoid conflicts with MAX IV Operating System naming conventions by choosing overlay module names with the fourth character of the name other than C or S.

4.6.5.2 Module Cataloging

All initial and overlay load modules of nonresident REX services must be cataloged to reside in the real file or disc partition with which the global logical file name SM is associated.

Because these modules will be loaded into any available page of Z-space when required, the modules must be coded to be self-relocating or must be cataloged as relocatable modules with an address bias of Ø.

Since such modules must occasionally execute privileged instructions, they must be cataloged PECULIAR PRIVILEGED when processed by the Task/Overlay Cataloger (TOC).

4.6.5.3 External References

The MAX IV Module Loader ML which is employed to load nonresident REX services contains no capability to permit EXTernally defined labels to be resolved at load time.

If a nonresident REX service needs to have such references resolved, a resident prefacing segment of code that can be link-edited during SYSGEN must be supplied. Such prefaces must be named in either the N$xxxx or the N$$xxx format and must reside in a file

that will be accessed by the link-edit phase of SYSGEN (for example, file UL). When employing a preface for a nonresident REX service that uses the standard MAX IV linkage convention, the contents of Bits 8-15 of Register 8 must be preserved.

Preface routines may invoke the allocation and loading process by a branch to the entry points stored in words UCBSRE or UCBCRE of the Upper Control Block when preface processing is completed.

All nonresident REX services may reference any system-defined Z-space operands whose addresses are defined in the System Equate File (IVSEQU) File without any action required by a preface routine.

4.6.5.4 Service Exit

When the nonresident REX service has completed execution, it must pass control to either UCBSRX or UCBCRX. The configuration of any nonresident REX service in the system causes both routines to be configured in the system.

If a REX service desires to pull from the Push Stack the registers that were pushed when the service was invoked, the service may exit using the UCBCRX routine with all registers containing the values to be present when control is returned to the invoker.

Similarly, a REX service which had Registers 1 through 15 on the stack upon initial entry, and does not wish to restore the registers itself, may exit using the UCBSRX routine.

If the nonresident REX pulled the two flag words from the front of the PS/PR stack, they must be pushed back before branching to the UCB vectors.

4.6.5.5 PS/PR Stack Size

The nonresident REX service linkage routines use the PS/PR Stack of a task to maintain certain information necessary to effect orderly transfers of control between the task and resident/nonresident REX services.

Consequently, in configurations of the MAX IV Operating System that use nonresident REX services, it is suggested that the size specified for task PS/PR Stacks during cataloging be double when the system includes only resident REX services.

4.6.5.6 Loading Overlays

If a nonresident REX service needs to load an overlay, it must do so by executing the following (or any equivalent-effect) sequence.

```
LDI,R15   @xxx      ...set characters 4-6 (xxx) of overlay name
BRU*      UCBNRO    ...invoke overlay loading
```

When transferring for overlay loading, the task PS/PR stack must be at the level initially present when the nonresident REX service was initially entered. Word UCBNRO is in the Upper Control Block is initialized during SYSGEN to the address of the routine R$LNRO. This routine not only loads the required overlay but also updates information in the PS/PR Stack to permit correct transfers of control.

4.6 5.7 Debugging Non-Resident Services

If you have written a custom REX service and its size is smaller than 256 words, then adding it to the system first as a nonresident REX service is good debugging practice. Thus, if any errors are found, a new version may be easily recataloged. Of course, one must re-boot the system and avoid calling the service being debugged while the new service is being prepared and cataloged. Performing a new system generation each time an error is encountered in the service is avoided.

If your custom REX service is larger than 256 words, then it may be easier to debug it by using the REX User trap service, #2F, and specifying the REX routine vector. It can then be broken down into 1-page overlays and cataloged as a nonresident REX service.

Debugging of the individual overlays of a nonresident REX service may be facilitated by setting the HL (Hold upon completion of load) system option for the calling task and ensuring that only one task calls the service being debugged. This will facilitate patching of the overlays or checking the interface arguments at the point of transfer to each overlay section.

4 7 PROGRAMMING CONSIDERATIONS - ALL REX SERVICES

4.7.1 USE OF MAIN PUSH STACK OF TASK

When a REX service subroutine needs small amounts of memory storage that is unique for the calling task, it may allocate/deallocate memory cells in the task's main push stack. A stack pointer table and stack area may be found in the TCB extension of every task. A pointer to the current task's stack pointer table is always maintained in the Lower Control Block by the Taskmaster, thus allowing information to be pushed/pulled to/from this stack by indirect reference of the table:

```
PSM*,R1  LCBSPT,7,15    ...allocate  5 cells, save R1-R7 in 1st 7
.
.
.
PLM*,R1  LCBSPT,15,15   ..restore all registers, deallocate space
```

This stack is initially sized when the task is installed For nonresident tasks this stack may be made automatically extensible by 256 additional words. However, due to the possibility that the extension space is not contiguous with the original stack space, it is forbidden to cross-index across separately allocated sections of the stack with a single index pointer.

4.7.2 USE OF THE SERVICE BUFFER OF THE TASK

Many REX services need a temporary scratch memory area in Z-space that is larger than is normally obtainable from a stack. The task's service buffer is often used for this purpose. In MAX IV, such a buffer is dynamically allocated. Before the buffer can be used, the following system subroutine must be called:

```
     BLM,R8  MM$GSB          ...get a service buffer for calling task
```

This subroutine is a memory management service used to allocate virtual space and actual memory for the service buffer. To use this buffer, the following instruction will load the address of the service buffer in register "z".

```
     LDM,y       LCBTCB      ...get TCB address
     LDM,z,y     TCBBUF      ...get service buffer address
```

The size of the service buffer (in positive integer format) is obtained as follows:

```
     LDM,a,y      TCBBUS     ...get size of buffer
```

This size will be 256 words for most tasks, but resident tasks may have a permanent fixed buffer smaller than this amount. This must therefore be checked in the general case. Use of the allocation routine MM$GSB will not affect a fixed service buffer because no new buffer will be allocated if one already exists.

When the service is finished using the service buffer, it must be deallocated as follows:

```
     BLM,R8   MM$PSB         ...Put (DEALLOCATE) Service Buffer
```

This subroutine is a memory management service used to dispose of the service buffer. If the virtual address of the buffer (in TCBBUF) is in the range of map zero pages known as the virtual space pool, the actual page will be deallocated and the virtual page returned to the pool If the task has a fixed buffer, it will not be deallocated by the subroutine.

4.8 NESTING OF REX SERVICES

REX services are naturally nestable If a REX service routine needs to call another REX service, it does so as the original invoker did.

The special stack is called the PS/PR stack and is used only for the purpose of saving PS and PR for each level of REX service nesting. The stack pointer table for this stack (the currently executing active task) is in word LCBPST not LCBSPT for the main stack .

This stack is needed for two reasons:

1. It avoids the overhead of having to manage REX LINK's in the main stack should nesting not be required.

2. It keeps all PS words of the task in a fixed place in a stack entry so that if the maps of the task are reassigned to other tasks, the Taskmaster can update all REX PS words to reflect new map assignments.

It is necessary to lock out the Taskmaster in order to manipulate PS and PR words in the stack. This is done automatically by routines and macros described later.

4.9 EXAMPLE OF A CUSTOM REX SERVICE

A hypothetical custom REX service called MARY will be implemented in the examples which follow. General narrative, describing how any custom REX service might be added, will be interspersed throughout these specific examples. The reader should be able to code custom REX services by using this training example as a guide. The next few sections are in the form of a specification for MARY.

4.9.1 SPECIFICATION OF MARY

CALLING SEQUENCE FOR MARY

CALL NUMBER. #44/REX,#32   CALL NAME: MARY
or
REX,#44 for MAX III compatible service

SERVICE TO BE PERFORMED BY MARY

MARY processes a specified quantity of data expressed in terms of four specified arguments and optionally performs some secondary operation on the data according to an option bit in the call. The call arguments are to be passed in four general registers in a MAX IV type of call and are to be passed as 6 inline memory cells following the REX instruction in the MAX-III-type call. All registers are restored to their initial states, except R11, which will contain a special data item upon return.

Calling Arguments for MAX-IV-Type Call

```
R8:  Service number and options with:
     Bit 0:     Optional process selector bit:
                ='1 optional conversion is to be performed.
                ='0 no optional conversion is to be performed.
     Bits 1-7:  Not used.
     Bits 8-15: Call number (=#44).

R12. Argument 1 is a virtual address of a table or buffer in
     calling program's O-space which contains data to be
     processed.
```

R13 Argument 2 is a number indicating the size of the table or buffer whose address is in R12.

R14: Argument 3 is a mask variable to be used in processing the data.

R15: Argument 4 is a default value to be used if some unusual data item is found during processing.

Calling Arguments for MAX-III-Type Call

All are inline memory arguments in calling program's I-space:

Word 0: REX call instruction and option (Bit 6, where:
= $2344 if option not specified.
= $23C4 if option is specified.

Word 1: Virtual address of table or buffer in calling program's O-space.

Word 2: A number indicating size of table or buffer whose address is in reword 1.

Word 3. A mask variable to be used in processing the data.

Word 4. A default data value to be used if some unusual data item is found during processing.

Word 5. An address of an error processing routine in I-space of calling task should service find errors that must be handled by the caller.

Word 6: The return point (next instruction to be executed) after service is executed if no errors are detected by service.

Arguments to be Performed by MAX IV Type Service

The contents of the data buffer, whose address and size were passed as calling arguments in R12 and R13, will be modified to contain the final processed data.

The 'N' condition code will be set (='1) if no errors are detected, and will be reset (='0) if errors exist. The program may test this condition code after calling the service and act accordingly.

All original registers are unchanged except for R11 which contains a special data item to be returned.

Arguments to be Returned by MAX III Type Service

The arguments for this service are the same as the MAX IV type service except the optional data item will be returned in R2 instead of R11.

Since no condition codes can be tested universally, an inline error address argument will be used as an alternate return address if errors are detected. The user is expected to have an error processing routine at this address unless it contains the address Ø, in which case the task will be aborted.

4.9.2 MAX IV INTERFACE TO MARY

The "heart" of MARY, and its MAX IV-type interface will be assembled in one module:

```
          INT    M$MARY,MARY
          EXT    I4$XIT                   REX SERVICE RETURN
          EXT    RX$SOO                   SELECT INVOKER'S O-MAP
 M$MARY   EQU    $        ENTRY-POINT    MAX IV TYPE ENTRY, R1-R15 IN
                                           STACK, R1 GONE
          BLM,R9          MARY           CALL HEART
```

This interface routine is simple to code since we are planning to make the subroutine MARY use exactly the same calling arguments as this MAX IV-type service call. Thus, the closed subroutine MARY may be called immediately upon entry to the service. Assume also that subroutine MARY will return its special data argument in R11 so that the interface need only provide a way of returning this argument to the calling program. The method chosen here is to store the argument in the main stack at a position where the original R11 of the calling program was saved upon entry:

```
   LDM*,R1  LCBSPT    ...get current stack pointer
   STS,R11,R1Ø        ...save new R11 in R11 save area - note,
                         R1-R15 stored in words Ø-14 of the stack.
```

It is now only necessary to restore all registers , release stack space, and return control to the original calling program:

```
     BRJ  I4$XIT      ...restore registers and return
```

4.9.3 THE HEART OF MARY

The subroutine MARY must now be coded so that it contains the "heart" of the service:

```
MARY EQU $ ...entry point
```

The arguments we must process are still in R8, R12-R15, and also in the main stack. We can choose to avoid these registers as working registers and reference the necessary arguments directly in the registers when they are needed, or we can destroy the arguments if we need these registers and reference them from the stack as needed. R1 is an exception, it is already destroyed by the REX entry and must be referenced from the stack where it is already stored, if needed.

If we need more temporary memory cells for working space - we can allocate them from the main stack.

```
      PSM*,R0  LCBSPT,1,10      ...allocate 10 cells, save only "R0"
      LDM*,R1  LCBSPT           ...load new stack pointer
```

In the case above, 10 cells are allocated and if the current stack pointer is loaded into register R1, then these cells can be individually referenced with short-displacement instructions. Because of the way MAX IV stacks can be automatically extended if they should overflow, you should not try to address the previous stack level where the registers were saved while this new push is active using the same index pointer. If stacks were not extensible, the previous stack level would be contiguous with the newly allocated cells and could be indexed with the new current stack pointer. However, since this can not be depended upon, it is recommended that it not be done.

If you need to alternately address sections of the stack that are allocated through separate push operations, the current stack pointer must be fetched into a unique index register at each push level while the stack pointer is still current:

```
      LDM*,R2  LCBSPT           ...get first level pointer
      PSM*,R7  LCBSPT,X,Y       ...go to new level
      LDM*,R3  LCBSPT           ...get new level pointer.
```

Another solution is offered below:

If you need all registers as work registers, then save the registers again at the second push level so that a single pointer in R1 can address call arguments and temporary cells:

```
      SFS,R12,0                 ...save R12-R15 in words (0-3)
      STS,R8,4                  ...save R8 in word  4)
```

Assume we must clear our 5 additional temporary cells as an initial condition:

```
      SURD,R14,R14              ...clear R14,R15
      SFS,R14,5                 ...clear 2 cells
      SFS,R14,7                 ...clear 2 more cells
      STS,R14,9                 ...clear 1 more cell
```

Notice that original arguments in R14,R15 have been destroyed, but may be reaccessed at any time from cells 2 and 3 of the current stack.

To process the data in the buffer, we must find a way of passing data from the task's O-space to the system's Z-space, and new data back to the task's O space. We will begin coding the processing loop below. The iterations of the loop are determined by the argument passed originally in R13 (and still there). This is the buffer size. Before we use this argument, ensure it is a valid loop count (positive, non zero).

```
       TPR,R13,R13              ...transfer to self, set CC.
       HZS,EROUT                ...exit to error routine if zero
       HNS,EROUT                ...exit to error routine if negative
LOOP   EQU $                    ...now, start the loop
```

To get each data item, we will use the buffer address still in R12, but remember that this address has no meaning in the Z-space in which we are currently executing. It only has meaning in the O-space of the calling task which we must select before fetching any arguments from it.

```
BLM,R9  RX$SOO            ...select REX invoker's O-Map
```

All operands that are accessed from now on come from calling tasks O-space:

```
LDX,R14,R12               ...get next word in data buffer of task
```

At this point we might be able to process all the data in general registers. We can continue to leave the task's operand space selected as long as no memory operand requests are made while processing the data in the general registers.

```
---)                      ...routine for data processing, makes
---)                      ...no memory operand requests
```

We can now store the new data, assumed in R15) back in the buffer:

```
STX,R15,R12               ...store word in buffer
```

Likewise, if all the words in the buffer can be processed in this manner, we could complete the entire loop with this O-space selected. The loop tally would be as follows:

```
ABR,R12,B15               ...add one to buffer address
SBR,R13,B15               ...subtract one from buffer size
HZR,LOOP                  ...repeat loop if still any data
```

Notice that an instruction such as "ABRB,R13,B15 LOOP" could be used just as well, because the branch address is not considered to be an operand in a CLASSIC, but as an instruction word referencing the I-space.

If processing of the data requires referencing of the stacks or any Z space arguments, the simple loop above will not suffice. Instead, as each word is fetched from the task's operand space:

```
BLM,R9  RX$SOO            ...select REX invoker's O-Map
LDX,R14,R12               ...get first (next) data item from
                             buffer in O-space
```

The MAP 0 operand space must now be reselected:

```
SZOM
```

Now we can process the data using Z-space arguments exclusively:

```
---)
---                       ...process data in registers
---)
```

```
     SFS,R14,5                  ...save temporary value in stack
     ---}
     ---}                       ...process more data
     ---}
     LFS,R14,5                  ...retrieve temporary value
     ADSM,R10,9                 ...accumulate "checksum" in stack
```

Fetch other map zero arguments:

```
     LDM,R5   LCBTCB            ...get TCB of calling task
     LFS,R14,R2                 ...get Z-space default arguments from stack
     ---}
     ---}                       . process more data
     ---]
```

When we are ready to return final data to the buffer we must reselect the task's O-space.

```
     BLM,R9   RX$SOO            ...select REX invoker's O-Map
     STX,R15,R12                ...store in task's O-space
     SZOM                       ...reselect MAP 0
```

NOTE: If the REX writer finds it necessary to get the REX return PS and select a map because it is not possible to call RX$SOO or RX$SIO, then the REX must lock out the Taskmaster for the duration of the time that the return PS is held in a register. Example:

```
     SIA,#F                     ...lockout Taskmaster
     REXFPS 8                   ...get task's return PS
     SOOM,R8                    ...select task's O-Map
     RIA,#F                     ...TASKMASTER may be unlocked
```

No PS word should ever be held in a register unless the Taskmaster is locked out because of the possibility of the Taskmaster reassigning the hardware map or register block of the task, and the Taskmaster has no way of knowing that a custom REX service had the old copy of the task's map assignments in some general register.

A compromise between staying in the task's operand space and having to reselect this space for every word retrieved and restored is possible if the O-space selection is partially pulled out of the loop. In this case, the O-space needs to be selected once for every loop iteration when the previous result is stored -- and when the next datum is fetched. The entire loop shown above is repeated below:

```
     ---}                       ...loop initialization
     ---
     BLM,R9   RX$SOO            ...select task's O-Map
LOOP EQU $
     LDX,R14,R12                ...get word from buffer to be processed
     ---}
     ---]                       ...process data in registers
     ---}
     SZOM                       ...MAP 0 arguments required
```

```
      SFS,R14,R5
      ---}
      ---}                  ...more register and Z-space processing
      ---}
      LFS,R14,R2            ...load mask & default arguments
      ADBM,R10,R9
       ...
      ---}                  ...process more data
       ..}
      BLM,R9  RX$800        ...select task's O-Map
      STX,R15,R12
      ABR,R12,B15
      SBRB,R13,B15   LOOP
      SZOM                  ...reselect MAP 0
```

Assume that an option bit of the service selects alternate processing routines above. The option bit 0 can be tested in the temporary stack (Word 4, where R8 was saved:

```
       TBSM,B0,R4
       HCS,ALTER
       ---          \
       ---           >----normal processing routine
       ...          /
       HOP,COMN
ALTER  ---          \
       ...           >----alternate processing routine
       ---          /
COMN   ...
```

The option bit 0 can also be tested in R8 if that register's contents have survived (which have not, in this case).

Assume that the special data item to be returned is a "checksum" that was accumulated in a stack reserve word. To return this argument, load it into R11.

```
      LDS,R11,R9            ...get checksum from temporary cell
```

We have now finished MARY's data processing except that we must return control to either the MAX IV type or MAX III type of interface. Before we return, we must release our temporary stack.

```
      PLM*,R0  LCBSPT,1,10
```

We will leave the job of pulling the final Register Save area of the stack for the appropriate interface.

If R9 the original link register) has survived through all this, we can immediately return control:

```
      BRX,R9
```

If R9 has been destroyed, we could have previously moved it to another register or saved it in the stack.

Everything is about complete, but there is still an error routine that we must code:

```
     EROUT  EQU $
```

Assume that the only thing necessary is to load a "dummy" return argument:

```
     ZRR,R11           ...default value of special return argument
```

And then reset the "N" condition code in REX LINK PS:

```
     LBR,R2,CCN
     REXRCC 2,3
     BRX,R9            ...return
     END               ...end assembly
```

4.9.4 MAX III COMPATIBLE INTERFACE TO MARY

```
       INT M$$MAR
       EXT MARY
M$$MAR EQU $        ENTRY POINT
```

This service is separately assembled so that it will not be linked into the MAX IV nucleus should MAX III type services not be optioned. The first job of the entry interface is to gather the inline memory arguments from the calling program's virtual instruction, space and load these into the registers that MARY expects them to be in upon being called.

```
     REXFPS   7,1        ...get virtual address of next
                            instruction after REX
     BLM,R9   RX$SIO     ...select task's IM as my OM
```

Note that at this point we have the virtual address of the REX instruction in the REX PR link (R7). We also have selected the I space map of the calling program as the map from which this REX service subroutine will fetch all subsequent operands. Also note that RX$SIO and RX$SOO contain the necessary lockouts to protect the PS while selecting the task's maps. They return to the caller unlocked. Now lets actually go fetch the REX instruction and four inline arguments:

```
     LPM,R11,R7     -1    ...get REX instruction (-1) and inline
                             args (0-3)
     SZOM                 ...reselect MAP zero
```

Note we have loaded R11-R15 with only the first four arguments of the five specified, using the pointer in R7 which is referenced to the calling program's I-space. Also, we reselect MAP 0 since no more operands need be fetched at this time. The fifth inline argument will be used in the exit interface if errors are detected in route. For now, we will increment the REX PR link by four to be in a better position to fetch this argument later:

```
LBR,R2,B13          ...increment REX Link PR by 4.
REXMRA   2,1
```

Now move the option bit of the REX instruction (BIT 8) to Bit 0 in R8 so MARY can look at it in the MAX IV way. The REX call number (#44) need not be constructed in R8 since the service does not reference it. Other spurious option bits need not be masked out since they are not referenced in the service subroutine.

```
MBL,R8,R11          ...move R11(bit 8) to R8 (bit 0)
```

Now MARY can be called as before.

```
BLM,R9    MARY
```

Upon return, MARY has returned the same argument in R11, but this is not the register this form of the service chose to return it, so first move it to the R2 entry in the register save stack.

```
LDM*,R1   LCBSPT       ...get register save address
STS,R2,R1              ...put R2 in stack R2 for return
```

If subroutine MARY had an error, we wish to know:

```
REXPPS   2,1           ...get REX return PS
TBR,R2,CCN             ...is error set in CNN of REX LINK PS
HCR,NOERR              ...hop if no error
```

The fifth inline argument must be fetched and used as a return point in the calling program if an error is detected. Note that we have already incremented LCBRPR by four so that the fifth argument will be the next one fetched.

```
REXFRA   6,1           ...get address of next inline  argument
BLM,R9   RXSSIO        ...select I-space as my  O-space
LDX,R7,R6              ...get error return address
SZOM                   ...return to Map 0
```

If the error routine address is zero, we assume that the caller has not specified an error return routine and will abort the task with a reason code "ZAP".

```
HZR,SETRET             ...skip if error routine address is
                          supplied.
REX,ABORT              ...terminate execution of calling task
DFC @ZAP               ...with this code
```

To effect a return to the alternate error recovery routine, store the new address in the REX LINK PR.

```
SETRET EQU     $
       REXSRA  7,1          ...new PR
       HOP,FINL             ...go to common exit below
```

If no errors were processed, we must skip the final inline argument.

```
   NOERR  LBR,R2,B15
          REIMRA    2,1     ...step over 5th argument
```

Now everybody can exit. We must now restore the registers the same way they were saved in the push stack.

```
      FINL BRU   I4$XIT     ...go back to task

      END
```

4.9.5 SYSGEN CONSIDERATIONS FOR MARY

We must consider how we are going to get MARY installed into a MAX IV system. A study of the MAX IV SYSGEN, System Guide Manual, will reveal that you must specify one or two statements.

The first statement:

IIISERVICES

will specify that for every MAX IV service to be optioned, either standard or user-coded, a MAX III compatible service will also be supplied. This is true only for service numbers less than $50.

The second statement.

SERVICE MARY,$44

will specify that MAX IV should add a service entry point with the external name M$MARY to the MAX IV service table. If MAX III-type services are also specified, it also specifies that a service entry point with the external name M$$MAR be added to the MAX III-type service table, also at position $44.

To get our services installed into the nucleus of MAX IV, it is only necessary to include the object modules of the assembled services in the UL (User's Library) file when the operating system elements are link-edited

CHAPTER 5
REX ENTRY AND EXIT MACROS AND SUBROUTINES

The REX Entry/Exit package is a revised method of saving and restoring REX links. The entry service will allow nesting of REX calls, and the exit service will contain utility routines for commonly performed functions. Those functions include restoring registers and setting condition codes for return to the invoking task

Upon a level four trap, the entry sequence in I4.UTR saves the return PR and PS on the tasks PR/PS stack That stack's control table is pointed to by ECBPST. After preparing a PR and PS appropriate to the REX, I4.UTR executes a CIR. I4.UTR sets condition codes in the return PS and modifies the return address for MAX III type REX calls.

The exit code of I4.XIT sets the active interrupt latch for level 4, restores the REX return PR and PS to the appropriate level 4 vectors and CIR to the invoking task

All 15 general registers are saved on the task's main push stack before REX entry for MAX III and MAX IV. The contents of R1 is unpredicatable upon REX entry, however, a copy exists on the stack.

5.1 USER INTERFACE

5.1.1 REX SERVICES

User written custom REX services will require modification to use the Revision F.0 or later REX Service subroutines. There is no compatibility possible between the old and new REX entry and exit procedures. The result of a combination of the two methods is unpredictable.

There is no change to the user interface for MODCOMP supplied routines. However, at the internal level, nested REX calls will no longer require the caller to save and restore the REX return vectors.

The user interface consists of a combination of REX service macros and subroutines. The macros will be defined in file IVMAC on the standard MAX IV MC file. The subroutines will be resident in the MAX IV nucleus

5.1.2 MACRO DEFINITIONS

To obtain the REX Service macro definition, use INS* MC,IVMAC This statement should replace the INS* MC,IVSEQU, if present and precede any local equates. The routines using this service must be privileged, MAP 0 routines.

Fetch REX Return PS

REXFPS   reg

The REX return PS will be placed in "reg," where "reg" is General Register 1 to 7. No other registers are affected.

Fetch REX Return Address

REXFRA   reg[,wrk]

The REX return address will be placed in "reg," where "reg" is General Register 1 to 7. If R1 is used, short instructions will be generated. If "reg" is 8 to 15, an index register "wrk" must be provided. No other registers are affected.

Fetch REX Return PS and PR

REXFRP   reg

The REX return PS/PR pair will be pulled from the stack pointed to by LCBPST into "reg" and "reg"+1, respectively. No other registers are affected.

Increment REX Return Address

REXIRA   [wrk

The REX return address will be incremented by one. "wrk" is an optional work register in the range of 1 to 7. If R1 is used, short instructions will be generated. If "wrk" is not specified, R1 will be saved and restored from the task stack pointed to by LCBSPT. No other registers are affected.

Modify REX Return Address

REXMRA   reg[,wrk]

The REX return address will be modified by the displacement contained in "reg." "wrk" is an optional work register in the range of 1 to 7. If R1 is used for "wrk," short instructions result If "wrk" is not specified, a save and restore of R1 will occur using the task stack pointed to by LCBSPT. No other registers are affected.

Reset REX Return Condition Codes

REXRCC   reg[,wrk]

The condition codes corresponding to bits set in "reg" will be reset in the REX return PS. "reg" will be masked to the low order four bits. "wrk" is an optional work register in the range of 1 to 7. If R1 is used, short instructions will be generated. If "wrk" is not specified, a save and restore of R1 will occur using the task stack pointed to by LCBSPT. No other registers are affected.

Set REX Return Condition Codes

REXSCC   reg[,wrk]

The condition codes corresponding to bits set in "reg" will be set in the REX return PS. "reg" and "wrk" are as in REXRCC. No other registers are affected.

Set New REX Return Address

REXSRA   reg[,wrk]

A new REX return address contained in "reg" replaces the old REX return address. "wrk" is as in REXIRA. No other registers are affected.

Store New REX Return PS and PR

REXSRP   reg

A new REX return PS/PR pair contained in "reg" and "reg" + 1, respectively, are pushed onto the REX return PS/PR stack pointed to by LCBPST. No other registers are affected.

Set New REX Return PS

REXSPS   reg[,wrk]

A new REX return PS contained in "reg" are stored over the old PS contained in the stack pointed to by LCBPST. "wrk" is an optional work register in the range of 1 to 7. If "wrk" is not specified, a save and restore of R1 occurs using the task stack pointed to by LCBSPT. No other registers are affected.

5.1.3 REX SERVICE SUBROUTINES

These subroutines are resident in MAP 0. To use them, the caller must be privileged and MAP 0 resident. The externals will be resolved by the linkage editor during SYSGEN.

REX Exit Service I4$CIR

Call: BRU I4$CIR

Service Performed:

Control will be returned to the invoker of the REX Service using the REX return PS/PR from the stack pointed to by LCBPST. No registers are changed.

REX Exit Service I4$ERR

```
Call:  LDI,R15   Retcode
       BRU       I4$ERR
```

Service Performed:

The N condition code in the return PS is reset. Bits 12-15 from R15 will be ORed to the condition codes in the REX return PS. R1-14 will be restored from the task stack pointed to by LCBSPT, while 15 words are released from that stack. R15 is returned intact. Control passes to I4$CIR to exit the REX.

REX Exit Service I4$ERX

```
Call:  LDI,R15   Retcode
       BRU       I4$ERX
```

Service Performed:

As in REX Exit Service I4$ERR, except that the R15 returned to the user is restored from the main stack pointed to by LCBSPT.

REX Exit Service I4$ESX

Call: BRU I4$ESX

Service Performed:

An event scan is requested. Control then passes to I4$XIT.

REX Exit Service I4$SCn

Call: BRU I4$SCn

Service Performed:

Condition codes are loaded according to the value of n. 'n' must be a decimal digit in the range of $0<=n<=7$. That is, a call to I4$SC4 will set 'NZOC to '0100]. A branch is then effected to I4$ERR.

REX Exit Service I4$XIT

Call: BRU I4$XIT

Service Performed:

This is the normal MAX IV REX Exit. Registers 1-15 are restored from the task stack pointed to by LCBSPT and control passes to I4$CIR for the REX exit.

REX Exit Service I4$X13

Call: BRU I4$X13

Service Performed:

Registers 1 through 13 are restored from stack and 15 entries are deleted from stack. A branch is then effected to I4$CIR.

REX Exit Service I4$X14

Call: BRU I4$X14

Service Performed:

Registers 1-14 are restored from stack and 15 entries are deleted from stack. A branch is then effected to I4$CIR.

Select REX Invoker's Operand Map

Call: BLM,R9 RX$SOO

Changed: R4

Service Performed:

The REX invoker's operand map is selected as the current operand map. This can enable operand fetches for REX calls.

Select REX Invoker's Instruction Map

Call: BLM,R9   RX$SIO

Changed: R4

Service Performed:

The REX invoker's instruction map is selected as the current operand map. This can enable parameter fetches for MAX III type REX calls.

Fetch REX Call and Argument

Call: BLM,R9   RX$RCA

Returns: C(R8) = REX Call Word
          C(R15) = First Word following REX

Changed: R4,R7

Service Performed.

The REX call word subsequent argument and REX invoker's PS are returned in the specified registers. This service may be used by MAX III type REX Services to prepare for entry to the MAX IV routine.

Format MAX IV Entry from MAX III REX (A)

Call: LDI,R8   NWORDS
          BLM,R9   RXSCKO

Returns: C(R8) = REX Call Word and Option Bit
          C(R10) = Address of REX Invoker's PS
          C(R15) = First Word following REX

Changed: R4,R6,R7

Service Performed

The REX return address is modified by the contents of R8 to bypass inline arguments. The return registers are set as above. Bit 0 of R8 will be set equal to Bit 8 of R8.

Format MAX IV Entry from MAX III REX (8,

```
Call:     LDI,R8    NWORDS
          BLM,R9    RX$ENR
```

```
Returns:  C(R8)  = REX Call Number and Option Bit
          C(R11) = Inline parameter #5
          C(R12) = Inline parameter #4
          C(R13) = Inline parameter #3
          C(R14) = Inline parameter #2
          C(R15) = Inline parameter #1
```

Changed. R1,R10

Service Performed:

The return registers are set as above. R8 contains only the REX number and bit 0 of R8 will be set equal to bit 8 of the REX call word. The REX return address will be modified by the displacement contained in R8 upon entry to TS$ENR.

MAX IV Exit RX$EX1

```
Call:     BRU   RX$EX1
```

Options: If R8, bit 2 is set, R1 will be moved to return register R15.

Service Performed:

R1 will be moved to the R15 save area according to option bit 2 in R8. An event scan will be requested and control passed to I4$XIT for a MAX IV REX exit. I4$XIT will restore all 15 registers from the save area and return to the REX invoker.

MAX IV Exit RS$EX2

```
Call:     LDI,R2    NWORDS
          LDI,R3    REX+OPTIONS
          BRU       RX$EX2
```

Service Performed:

RX$EX2 will reformat a MAX III style REX call with up to 5 inline arguments into a MAX IV REX call.

```
MAX III:  REX,number
              DFC     ERROR
              DFC     PARAM1
              DFC     PARAM2
              DFC     PARAM3
              DFC     PARAM4
```

```
Resultant MAX IV:
          LDI,R10    ERROR
          LDI,R11    PARAM4
          LDI,R12    PARAM3
          LDI,R13    PARAM2
          LDI,R14    PARAM1
          ZRR,R15
          REX,MAXIV
```

On return from the MAX IV REX service, if successful, the RETTCB option is checked (bit 0 of R0 , and if set, the TCB address is returned in R2. If the MAX IV REX was not successful, condition code N reset, then RX$EX2 will return to the user supplied ERROR address. If none was supplied, the REX invoker will be aborted with code ZAP.

5.2 SUMMARY OF MACROS AND SUBROUTINES

5.2.1 MACROS

REXFPS	Fetch REX Return PS Link
REXFRA	Fetch REX Return PR Link
REXFRP	Pull REX Return PSD from PS/PR Stack
REXIRA	Increment REX Return PR Link by 1
REXMRA	Modify REX Return PR by a Displacement
REXRCC	Reset REX Return PS Condition Codes
REXSCC	Set REX Return PS Condition Codes
REXSRA	Store New REX Return PR Link
REXSRP	Push New REX Return PSD onto PS/PR Stack
REXSPS	Store New REX Return PS Link

5.2.2 SUBROUTINES

I4$CIR	REX Exit Service; No Register Restore
I4$ERR	REX Exit Service; Set Return Codes
I4$ERX	REX Exit Service; Set Return Codes
I4$ESX	REX Exit Service; Event Scan
I4$SCn	REX Exit Service
I4$XIT	REX Exit Service
I4$X13	REX Exit Service
I4$X14	REX Exit Service
RX$SOO	Select REX Invoke's O-MAP
RX$SIO	Select REX Invoke's I-MAP
RX$RCA	Fetch REX Call Word and Argument
RX$CKO	Format MAX IV REX Entry from MAX III REX
RX$ENR	Format MAX IV REX Entry from MAX III REX
RX$EX1	MAX IV RETTCB Option REX Service Exit
RX$EX2	Format MAX IV REX Entry from MAX III REX

APPENDIX A
NUMERIC LIST OF MAX IV EXECUTIVE SERVICES

MAX III COMPATIBLE RPX SERVICES, for example, $23xx, where xx is service number. Refer to the MAX III GENERAL OPERATING SYSTEM Reference Manual for detailed description.

NUMBER	SERVICE
0	READ (input data from device to memory)
1	WRITE (output data to device from memory)
2	REWIND (reposition device to Beginning-of-Media)
3	BACKSPACE FILE (search reverse until file mark or Beginning-of-Media detected.
4	BACKSPACE RECORD (space reverse over one physical record)
5	ADVANCE RECORD (space over one physical record)
6	ADVANCE FILE (search until file mark or End-of-Media detected)
7	WEOF (write EOF mark)
8	HOME (normalize device and perform device-dependent initialization
9	TERMINATE (abnormally) - (dequeue and terminate task-caused or all I/O device operations)
$A	ASSIGN (File = DEVICE, FILE = FILE, or OPEN transient FILE)
$B	TEST ASSIGNMENT (deliver file/device assignment information)
$C	IOWAIT (wait for complet on of one or more I/O operations)
$D	RESERVED
$E	MAX IV FILE MANAGER CALLS (not implemented in MAX III)
$F	FORTRAN RUN-TIME ENTRY SERVICE - optional
$10	MESSAGE/HOLD (notify operator and suspend (optional) until operator action)
$11*	WAIT/HOLD (suspend indefinitely or until operator action,
$12*	EXIT (normal task termination)
$13*	ABORT (abnormal task termination)
$14	DELAY (suspend for finite time)
$15	RESUME continue execution of another suspended task)
$16*	ACTIVATE (start execution of another task at specified priority level)
$17*	KILL (abort execution of another task)
$18*	CONNECT (schedule interrupt or timer for delayed RESUME, ACTIVATE, or KILL of a task at a specified priority level)
$19*	UNCONNECT (unschedule and release interrupt or timer)
$1A*	THAW (enable an interrupt or timer that has been CONNECTED)
$1B*	FREEZE disable and hold interrupt or timer that has been CONNECTED)
$1C	CHANGE (change priority task)

* Asterisk denotes reserved or privileged services.

$10*	RELINQUISH TILL NEXT EVENT (force context switch/give time to lower priorities
$1E*	RELINQUISH TILL CONDITION SATISFIED (pause until some task or interrupt sets or resets a specified bit or changes a value)
$1F	DEBUG (reserved for DEBUG processor usage and pause directive)
$20	FLOATING POINT OVERFLOW
$21	MODACS III
$22	RESERVED
$23	TAKE (take exclusive use of assigned device)
$24	GIVE (give up exclusive use of assigned device)
$25	RESERVED
$26	RESERVED
$27	ESTABLISH (make a nonresident task resident semi-permanently
$28*	DEESTABLISH (return task to nonresident state)
$29	ALLOCATE (allocate blocks of transient memory)
$2A	DEALLOCATE (return blocks of transient memory to free-space pool)
$2B	LOAD-CHAIN (load overlay on top of calling task)
$2C	(--- LINK-LOAD --- not implemented)
$2D	LOAD-OVERLAY (load overlay in buffer for later use)
$2E	(--- CALL --- not implemented)
$30	PAUSE (reserved for pause directive, not implemented in MAX III)
$31	(--- PULL --- not implemented)
	(--- PUSH --- not implemented)
$32	MAX IV Executive Service calls, not implemented in MAX III) (--- IOX --- not implemented)
$33	Unassigned
$34	GET RAW PARAMETER (get next parameter in byte string between delimiters)
$35	COLLECT AND CLASSIFY PARAMETER (get next parameter and determine if numerical)
$36	(--- MATCH --- not implemented)
$37	ASCII WORD PARAMETER-TO-CAN-CODE (convert 3-byte string to CAN integer)
$38	ASCII NUMBER PARAMETER-TO-BINARY (convert byte string to double binary integer)
$39	CAN CODE-TO-ASCII (conversion to byte string of three characters)
$3A	INTEGER-TO-DECIMAL ASCII (conversion to byte string of digits with trailing blanks or leading blanks
$3B	INTEGER TO HEXADECIMAL ASCII (conversion to byte string of hexadecimal digits)
$3C	DOUBLE INTEGER-TO-DECIMAL ASCII (conversion to byte string of digits with decimal point with leading or trailing blanks)
$3D	RESERVED
$3E	ROLL task into or out of main memory

```
#3F          DUMP CORE (hexadecimal dump of core block to printing
               device)
#40          TIME-OF-DAY (snapshot of current year, month, days,
               minutes, and ticks
#41          Initialize load module file resident directory
#42          RESERVED
#43          TASK INFORMATION (give snapshots of critical task
               control variables)
#44 \
#45   --     ---Reserved for MAXNET---
#46 /
#47 \
 .
 .    --     For user-coded global REX Services
 .
#4F /

#50 \
 .
 .    --     For user-coded local REX Services
 .
#7F /
```

MAX IV ONLY EXECUTIVE SERVICES, for example, Ø2312, where the right byte of register Ø contains the Service Number.

NUMBER	SERVICE
Ø	READ (read one physical record from I/O device)
1	WRITE (write one physical record to I/O device)
2	REWIND (rewind I/O device)
3	BKFILE (backspace file on I/O device)
4	BKRECORD (backspace record on I/O device)
5	AVRECORD (advance record on I/O device)
6	AVFILE (advance file on I/O device)
7	WEOF (write End-of-File mark on I/O device)
8	HOME (normalize I/O device)
9	TERMINATE (terminate I/O device)
ØA	ASSIGN (assign logical file to I/O device or another logical file)
ØB	TASSIGN (test assignment of logical file and interrogate its assigned device)
ØC	IOWAIT (wait for completion of one or more I/O operations)
ØD	RESERVED
ØE	RESERVED
ØF	FORTRAN RUN-TIME ENTRY SERVICE - optional
Ø1Ø	MESSAGE/HOLD (print message to operator)
Ø11	WAIT/HOLD (suspend calling task indefinitely)
Ø12	EXIT (terminate execution of calling task - normally
Ø13	ABORT (terminate execution of calling task - abnormally)
Ø14	DELAY (set or get local delay timer status
Ø15	RESUME (resume execution of a suspended task)
Ø16	ACTIVATE (start execution of a task
Ø17	KILL (abnormally terminate another task)
Ø18	CONNECT (allocate timer or interrupt to schedule task control function)
Ø19	UNCONNECT (disable and deallocate task scheduler)
Ø1A	THAW (enable allocated task scheduler - timer or interrupt)
Ø1B	FREEZE (disable connected timer or interrupt)
Ø1C	CHANGE (change priority of specified task,
Ø1D	RELINQUISH (let lower priority tasks use CPU till next system event or let higher priority tasks know calling task has caused system event)
Ø1E	RELINQUISH TILL CONDITION SATISFIED (pause until some task or interrupt sets or resets a specified bit or changes a value,
Ø1F	DEBUG (reserved for DEBUG processor usage and pause directive)
Ø2Ø	STATE (set or get JOB/task states of calling task)
Ø21	MODACS III
Ø22	WORKBENCH SERVICES
Ø23	TAKE (take exclusive use of I/O device)
Ø24	GIVE (give up exclusive use of I/O device)
Ø25	RESERVED
Ø26	RESERVED

#27	ESTABLISH (establish the residency of specified non-resident task
#28	DEESTABLISH (make established task no longer resident)
#29	ALLOCATE (allocate region of private memory for calling task)
#2A	DEALLOCATE (deallocate region of memory for calling task)
#2B	LOAD-CHAIN (load overlay on top of calling task
#2C	RESERVED
#2D	LOAD-OVERLAY (load specified overlay program,
#2E	RESERVED
#2F	USERTRAP (cause calling task to trap locally on subsequent violation)
#30	PAUSE (reserved for pause directive)
#31	RESERVED
#32	RESERVED
#33	RESERVED
#34	GETPAR (parse next simple parameter in character string)
#35	COLLECT (parse next numerical or simple parameter in character string)
#36	RESERVED
#37	ATCAN (convert ASCII string to CAN-code)
#38	ATN (convert encoded ASCII string to binary number)
#39	CTA (convert binary CAN-code into ASCII string)
#3A	BTD (convert single binary number to decimal ASCII string
#3B	HEX convert binary number to hexadecimal ASCII string)
#3C	DTD (convert double integer to decimal ASCII string with decimal point inserted)
#3D	EVENT LOGGING
#3E	ROLL task into or out of main memory
#3F	DUMP (dump regions of memory in printed format
#40	GETSYS (get current time values or system information)
#41	Initialize load module file resident directory
#42	INTERTASK COMMUNICATION
#43	GETASK (get information about specified or calling task)
#44 – #46	--- reserved for MAXNET ---
#47 – #4F	For user-coded global REX services

#50	MODOPTION (modify system options of specified task)
#51	GETOPTION (get system options of specified task)
#52	MODPOP (modify program options of specified task)
#53	GETPOP (get program options of specified task)
#54	SETVAR (set named variable of specified task)
#55	GETVAR (get variable of specified task)
#56	DELVAR (delete variable of specified task)
#57	MARS (modify access rights in calling task)
#58	CREPRIVATE (create shared region of memory from private region in calling program)
#59	INSGLO (insert global shared region of memory)
#5A	INSPRI (insert private shared region of memory)
#5B	INSHLO (insert shared load module into memory)
#5C	EXTSHA (extract shared regions of memory)

```
#5D \
 .
 .   |--     For user-coded local REX services
 .
#FF /
```

APPENDIX B
USER'S FILE TABLE (UFT) FOR MAX-IV-ONLY SERVICES

The UFT is an 8- or 10-word table in the user's program that contains entry and return arguments associated with the REX service for standard device I/O operations. Process I/O devices require two additional arguments in the UFT (see descriptions of these devices).

Before an I/O REX service is performed, prepare 4 words in this table (as entry arguments) that contain the name of the logical file, specifications that describe the format and options desired, and buffer mapping information.

After an I/O REX service is performed, the I/O system processes and returns information in the other words of this table (return arguments). If Quick Return Mode is selected a UFT BUSY flag in the UFT STATUS WORD (Word 0) is reset to indicate completion. Thus, a UFT must be unique for the "life" of each queued I/O operation. To ignore the UFT BUSY flag, set the IGNORE UFT BUSY STATUS (IUS) flag in the UFT EXTENDED OPTIONS (Word 6).

A typical UFT would be coded as follows in a user's Assembly-language program. The FORTRAN Run-Time Package synthesizes a UFT in a labeled common block so the user need not be concerned with this table. Other FORTRAN extensions allow the user to deal with a UFT directly in the FORTRAN Program.

Further details for the use of the UFT may be found in the MAX IV Basic I/O System, System Guide Manual listed in the Preface.

Word	Operation	Set By	Purpose
0 (0th)	DFC 0	BIOS	Status after operation
1 (1st)	DFC @XYZ	User	Logical File Name
2 .	DFC #A000	User	Formats and Options
3 .	DFC 0	User or BIOS	Device Position Index
4 .	DFC 0	BIOS	Byte Count
5 .	DFC 0	BIOS	Pointer to Assign List
6 .	DFC 0	User	Extended Options
7 .	DFC 0	User	Mapping Information
[8].	[DFC BUF]	User	Buffer Address (chain)
[9].	[DFC CNT]	User	Buffer Size

To use R14/15 for the buffer address and size, make Word 6 of the UFT "DFC #4000" and do not use Words 8 and 9.

<u>Word 0 (Status)</u>

Word 0 is set to zero by the I/O system after the REX service is called by the user. After the requested operation is performed, bits are set to indicate the error or event conditions that exist for the physical device to which the file is assigned. The user's program may test bits in this word after the execution of the service and make decisions concerning the results. Recovery from certain errors is attempted automatically by the system, if the

user requests "system recovery" in option Word 2 (explained below). This word has an "all inclusive" busy status bit (15) that, if set, means that the UFT is busy. It is set from the time an operation is queued until the operation is completed. If a UFT has Bit 15 set when an I/O REX Service is called, the I/O system assumes this UFT is "busy" from a previous I/O operation and relinquishes time until the operation is completed. This can be inhibited by a special option bit (in Word 6) that causes the I/O system to unconditionally queue the operation.

Word 1 (File Name)

Word 1, set by the user, is a call argument: the 3-character name of the logical file. This name must be expressed by a CAN (compress-alphanumeric code) data constant, and it must be a valid name in the task's File Assign Table or the Global File Assign Table.

Word 2 (Options)

Word 2, set by the user, is a call argument that describes the data mode and options to be used when the I/O operation is performed. A detailed schematic of this word is illustrated in the MAX IV BASIC I/O SYSTEM, System Guide Manual listed in the Preface. Word 6 is an extension of these options.

Word 3 (Position Index)

Word 3 specifies either a 16-bit position index or the address of a 32-bit position index. The default is a 16 bit index in this word. If the 32-bit position index is selected (Bit 4 in option word 6), this word contains the address of a doubleword memory location that is to be used as the position index. This option is useful when a disc partition greater than 16K sectors is to be addressed in random mode. (In sequential mode, a 32-bit position index is automatically maintained by the system.) For privileged tasks, other addressing parameters are available to address disc devices and will be discussed later. The position index is normally set to zero by the I/O system each time a REWIND function is performed, or when the file is initially "assigned". This word is incremented for each read/write or forward positional operation and decremented for each reverse positional operation. It keeps a count of the number of physical records encountered. For special devices such as the disc, which may be randomly addressed, this word may also be set by the user as a call argument to the "relative" address sector) of the random physical record to be used. When this sector address is not changed by the user, disc partitions act as "sequential" devices. So a randomly addressable device may be sequentially or randomly addressed. The random function is ignored for sequential devices. Whenever the random bit is set, the Position Index, as used by the device handler, is incremented upon successful completion of the operation, and is returned to the UFT and to a similar cell in the logical file's entry in the File Assignment Table.

Word 4 (Byte Count)

Word 4 is set to zero by the I/O system at the beginning of an I/O operation. After a read or write operation is performed, this word is set by the system to the number of bytes actually transferred by the assigned device (non-data chaining only). Regardless of the number of bytes specified by the user, this byte count does not exceed the size of the device's physical record (if fixed record size, or the user's specified data buffer, whichever is smaller. If the number of bytes in a physical record is greater than the specified buffer size during standard binary read operations to byte-type devices (for example, paper tape readers and card readers, the overflow/underflow (OVF) error bit and the (EOL) bit are set in the UFT status word, but the error (ER) bit is not set. All standard DMP devices have an even number of bytes-per-record transmitted except when certain device-dependent characters are detected in the data stream.

Word 5 (Assign List Pointer)

Word 5 is set by the I/O system and aids in a low-overhead response to each file's I/O operation. The contents of the word are determined after the first file operation using each UFT is performed, and then contains a value that links the file name to its entry in the File Assign Table or the Global File Assign Table so, each subsequent I/O operation using this UFT can be performed without a sequential search of the appropriate File Assign Table. The user need not be concerned with the contents of this word. This pointer is set by any file operation (for example, HOME, ASSIGN, REWIND) that uses a UFT for calling arguments.

Word 6 (Extended Options)

Word 6 specifies other characteristics about an I/O operation not available in Word 2.

Word 7 (Map Image)

Word 7 provides the I/O system mapping information about the data transfer. Normally, the operand map is the map used for data transfers and is selected by this word being zero. If the instruction map is to be used to resolve the virtual-to-real addressing during the transfer, this word is set to -1 (#FFFF). If the calling task is a one map task, these two parameters are identical. For privileged tasks, this word may specify an actual page number that contains a map image to be used for the transfer. This is useful to transfer data into another task's memory, and to initialize some memory not mapped into the calling task's addressing space. So privileged tasks may use this feature to perform I/O operations into actual memory regions not mapped by a hardware map but which are fully described in a "map-image" table that the system or the task has constructed to describe the contiguity of such actual memory regions.

Word 8 (Buffer Address or Chain Address)

Word 8 may be specified in R14, in which case it may be eliminated from the UFT if Bit 1 of Word 6 is set to specify that the buffer address and byte count are contained in Registers 14 and 15. This word contains the address of the programmer's I/O buffer in the map selected by Word 7. If data chaining is desired, this word contains the address of the chain list in the map selected by Word 9. Data chaining is usable only for Direct Memory Processor (DMP) devices.

Word 9 (Buffer Size in Bytes or Chain Control)

Word 9 may be specified in R15, in which case it may be eliminated from the UFT if Bit 1 of Word 6 is set to specify that the buffer address and byte count are contained in Registers 14 and 15. This word contains the byte count for non-chaining operations. Specify the maximum number of bytes desired, as a positive number. If data chaining is desired, this word contains mapping information for the chain list address given in Word 8. The chain list may be either in MAP 0 or in the map used for the data transfer (specified by Word 7). To indicate this, Word 9 contains -1 (#FFFF) or -2 (#FFFE) respectively. Any other negative value in this word is a parameter error.

MORE INFORMATION ON UFT AND READ

Refer to the MAX IV BASIC Input/Output SYSTEM, System Guide Manual (listed in the Preface for more information about the UFT and the Basic I/O data transfer modes or access methods

APPENDIX C
FLOW CHART OF UNIMPLEMENTED INSTRUCTION TRAP SUBROUTINE

The actual coding of this subroutine may be found in the source listing of the MAX IV SYSTEM ELEMENT LIBRARY in the module named "I4.UTR".

```
           .---------.
            \I$UTRP/     trap at hardware
             \    /---   interrupt level 4
              \  /       which selects
               \/        MAP 0 and RB=#F
                |
                V
   .---------------------------.     .---------------------------.
  /                             \    |SYSTEM PRIVILEGED ELEMENT   |
 /  ARE ANY OTHER INTERRUPT      \YES|  HAS FAILED--HALT TO       |
 \   LEVELS ACTIVE THAN 4?       /--->|  PREVENT FURTHER          |
  \                             /    |    DEGRADATION             |
   '---------------------------'     '---------------------------'
                | NO
                V
   .---------------------------.
   |FETCH 'PS' & 'PR' OF PRO-  |
   |GRAM THAT WAS TRAPPED-FROM |
   | DEDICATED CELLS (#20 &    |
   |#49) OF LEVEL 4 INTERRUPT  |
   '---------------------------'
                |
                V
   .---------------------------.
   |CONVERT THIS 'PS/PR' INTO  |
   |   REX LINK WHICH WILL     |
   |CAUSE EVENTUAL RETURN TO   |
   |CALLING PROGRAM FOLLOWING  |
   |   TRAPPED INSTRUCTION     |
   |  (PR+1).  PRESET CC TO    |
   |      NZOC = 1000          |
   '---------------------------'
                |
                V
   .---------------------------.
   |SELECT I-SPACE OF CALLING  |
   |PROGRAM -- FETCH TRAPPED   |
   |    INSTRUCTION WORD       |
   '---------------------------'
                |
                V
   .                           .
      FORMULATE A 'PS' FOR
   SERVICE ENTRY IN Z-SPACE
   '                           '
                |
                V
```

FLOW CHART OF UNIMPLEMENTED INSTRUCTION TRAP SUBROUTINE (CONTINUED)

WAS TRAPPED INSTRUCTION A REX,#32?

YES

DROP FROM LEVEL 4 TO CALLING TASK LEVEL; RESTORE R8 & CC OF CALLING TASK

SAVE 15 GENERAL REGISTERS IN MAIN PUSH STACK OF CALLING TASK

USE LOWER 7 BITS OF R8 AS INDEX TO ENTRY-POINT OF DESIRED MAX IV TYPE REX SERVICE SUBROUTINE

NO

4 - #F

WAS TRAPPED INSTRUCTION OP-CODE ANY OTHER REX INSTRUCTION?

NO

USE BITS 0-2 OF OP CODE AS INDEX TO ENTRY-POINT OF INSTRUCTION SIMULATION SUBROUTINE FOR 1X, 3X, 5X OR 'OTHER' OP-CODE

YES

USE LOWER 7-BITS OF OP-CODE AS INDEX TO ENTRY-POINT OF DESIRED MAX III TYPE REX SERVICE SUBROUTINE

DROP FROM LEVEL 4 TO CALLING TASK LEVEL, RESTORE R8 & CC OF CALLING TASK WITH FORMULATED PS, SAVE 15 GENERAL REGISTERS IN MAIN PUSH STACK AND ENTER SUBROUTINE SELECTED BY INDEX ABOVE

TO SELECTED M$$XXX OR SIMULAT.

XXX is name of a MAX III REX service subroutine

TO SELECTED M$XXXX

XXXX is name of a MAX IV REX service subroutine

INDEX

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 3624 FT. LAUDERDALE, FL 33309

POSTAGE WILL BE PAID BY ADDRESSEE

MODULAR COMPUTER SYSTEMS
1650 W. McNAB ROAD
P.O. BOX 6099
FT. LAUDERDALE, FLORIDA 33310

A ntion: TECHNICAL PUBLICATIONS, M.S. #85

Fold

MODCOMP

Please comment on the publication's completeness, accuracy, and readability. We also appreciate any general suggestions you may have to improve this publication.

If you found errors in this publication, please specify the page number or include a copy of the page with your remarks.

Your comments will be promptly investigated and appropriate action will be taken. If you require a written answer, please check the box and include your address below.

☐

Comments: ______________________________

Manual Title ______________________________

Manual Order Number ____________________ Issue Date ____________

Name ____________________ Position ____________

Company ______________________________

Address ______________________________

______________________________ Telephone (   ) ____________

[illegible]

MODCOMP

Corporate Headquarters:
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 6099, Ft. Lauderdale, FL 33310. Tel: (305) 974-1380, TWX: 310-372-7837

International Headquarters:
MODULAR COMPUTER SERVICES, Inc., The Business Centre, Molly Millars Lane, Wokingham, Berkshire, RG11 2JQ UK. Tel: 0734-786808, TLX: 851849149

Latin American Sales Headquarters:
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 6099, Ft. Lauderdale, FL 33310, Tel: (305) 975-6562, TLX: 3727852

Canadian Headquarters:
MODCOMP Canada, Ltd., 400 Matheson Blvd. East, Unit 24, Mississauga, Ontario, Canada L4Z 1N8, Tel: (416) 890-0666, TELEX: 06-961279

SALES & SERVICE LOCATIONS THROUGHOUT THE WORLD